

METADroid: Lightweight Android classification using meta-data

by

Yan Li

Bachelor of Computer Science, UNB, 2016

**A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF**

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor(s): Natalia Stakhanova, PhD/CS
Examining Board: Paul Cook, PhD/CS, Chair
Michael Fleming, PhD/CS, Reader
Howard Li, PhD/ECE, External Reader

This thesis is accepted

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

September, 2016

©Yan Li, 2016

Abstract

The Android system is the most widely used mobile system in the world and the user requirement is still increasing. According to [37], Android dominated the market with a 79.8% in 2013 Q2, 84.8% in 2014 Q2 and 82.8% share in 2015 Q2. Compared to other mobile systems, Android dominates most of the market and has the largest number of mobile users. Based on the research work [44] from PulseSecure.net, the number of new Android malware samples have been dramatically increased from 3809 in 2011, 214327 in 2012, 1192035 in 2013, 1548129 in 2014 and in the 2015 Q1 the new malicious sample is 440267. The approximate total value during the entire year of 2015 was a greater total than 2014. With that in mind, malware has always been the most pressing concern for the mobile application market. There are a number of analysis tools and architectures used for malware detection including static analysis, dynamic analysis, sandbox analysis and manually dissection. Consequently, all of the analysis approaches are time consuming. In order to effectively use our limited time and human resources, we present a lightweight pre-filtering tool(METADroid) which could be used to pre-classify the apps before the more expensive traditional static, dynamic analysis. The system includes whole 381 features and we analyzed their effectiveness for triaging. It will give you feedback of each apk in a short period of time and provide valuable prediction. Our experiments based on more than 158000 Android Applications collected from 8 markets around the globe.

Dedication

To my mother, for her unconditional love and support.

Acknowledgements

First I would give my deepest appreciation to my supervisor Natalia Stakhanova. For her genuine apprehension, encouragement, patience, and guidance, whose expertise and knowledge were generously shared.

I would like to thank Hugo Gonzalez, Andi Fitriah, Kamran Morovati and Alzahrani Abdullah for their technique assistance, selfless helpfulness and patience.

In addition, I would say thank you to all members in my ISCX lab, you all helped me get through the obstacles in my school work.

I would also like to thank my mother. Hua Yan, who supported me and always kept a positive attitude. At last I would like to give my great appreciation to several special people I met during this time, the elders and the sisters from Mormon, thank you all for your inspiration and selfless guidance on my life. Thank you for your support and blessings.

Table of Contents

Abstract	ii
Dedication	iii
Acknowledgments	iv
Table of Contents	v
List of Tables	viii
List of Figures	ix
Abbreviations	x
1 Introduction	1
2 Related work	5
2.1 Background on Android environment	5
2.2 Static analysis	8
2.3 Dynamic analysis	10
2.4 Metadata	11
3 Design	16
4 Features	20
4.1 Size	21

4.2	Time-stamp	23
4.3	Reputation	25
4.4	Third-party libraries and AndroidManifest file analysis	27
4.5	Certification analysis	34
4.6	Tool analysis	35
5	Dataset	36
5.1	Dataset for reputation establishment	38
6	Experiment setup	42
6.1	Implementation process	42
6.2	Experiment setup	48
6.3	Experiment results	49
6.3.1	Assessment of feature group individually	50
6.3.2	Classification based on combination of features	55
6.3.3	Chain classification	57
6.3.4	Experiments with Drebin dataset	59
6.3.5	Comparison with another static analysis tool	60
6.3.6	Performance analysis	62
7	Discussion	64
8	Conclusion and Future work	68
8.1	Conclusion	68
8.2	Future work	69
	Bibliography	78
A	Tables in feature chapter	79
A.1	Size features section	79

A.2	Timestamp features section	82
A.3	Manifest features section	117
B	Figures for structures of apps made by different tools	136
	Glossary	140
	Vita	

List of Tables

2.1	Existing studies that employ meta-data features in analysis	12
4.1	meta-data feature groups	22
4.2	Dex file features list	23
4.3	Timestamp features related to signature files	25
4.4	The list of features based on package name and author reputation . .	27
4.5	AndroidManifest elements counting features	31
4.6	Priority features	33
4.7	Permission features	33
5.1	Dataset app distribution	37
5.2	Details of app distribution in reputation dataset and experiment dataset	
	38	
6.1	Size based features classification based on different machine learning	
	algorithms	51
6.2	Triaging with only permission-based features	54
6.3	Detailed classification based on reputation features	55
6.4	Detailed classification based on tools features	55
6.5	Benign ratio of apps made by different tools	56
6.6	Triaging details	58
6.7	Time overhead with number of features included	63
7.1	Detailed classification based on tools features	64

A.1	Table of size feature group	79
A.2	Table of timestamp feature group part 1	82
A.3	TSable of timestamp feature group part 2	89
A.4	Table of timestamp feature group part 2	99
A.6	Table of manifest feature group	117
A.5	Ratio of files in the assets folder	135

List of Figures

1.1	Smartphone OS Market Share(Android is the highest all the time) . .	2
1.2	Number of available apps on Google Play market [56]	3
2.1	APK file format [29]	6
3.1	Overview of our design	17
4.1	A list of feature employed in the study by Teufl et al. [59].	21
4.2	Sample activity database from analysis	26
4.3	An example of legitimate elements in AndroidManifest file	28
4.4	An example of Google ads elements in AndroidManifest file	30
4.5	An example of AndroidManifest.xml file showing abnormally high priority value	32
4.6	Information about issuer and owner	34
4.7	Create a keystore to sign APKs	34
5.1	Sample APK signature	39
5.2	Distribution of apps by developers	40

6.1	Structure for apps signed by keytool	44
6.2	Structure for apps signed by jarsigner	45
6.3	Classification based on selected size features	51
6.4	Classification based on selected timestamp features	52
6.5	Classification based on selected features from AndroidManifest file . .	53
6.6	Classification based on selected combination features	56
6.7	Chain classification model 1	57
6.8	Chain classification model 2	58
B.1	Android Studio & Eclipse app zipinfo top lines	137
B.2	Android Studio & Eclipse app zipinfo bottom lines	137
B.3	Adobe air APKs zipinfo top lines	138
B.4	Adobe air APKs zipinfo bottom lines	138
B.5	Dot42 APKs zipinfo partial lines	139
B.6	Unknown tool	139

List of Symbols, Nomenclature or Abbreviations

Start writing here.

\sum \sum
 \bigcap \bigcap

Chapter 1

Introduction

Malware is short for *malicious software*. This is a brief description about malware given by Microsoft: “Malware is short for malicious software and is typically used as a catch-all term to refer to any software designed to cause damage to a single computer, server, or computer network, whether its a virus, spyware” [42]. Although it is a broad term that refers to a variety of malicious programs, there are several common types of malware: adware, bots, bugs, rootkits, spyware, trojan horses, viruses and worms.

Malware can reside on any operating system(OS), server, personal computer(PC), mobile device or any other kind of digital wearable device. In our work, we focus on the Android mobile platform. Over the past decade a number of open systems were developed for mobile platforms, Android is by far the most popular one according to an IDS report (Figure 1.1).

“In 2015, Android dominated the market with an 82.8% share. With more than an 80% domination of the smartphone market, the risk of malware infection is also dramatically increasing.” A study carried out by Rene Millman[43], found that 97% of malware focuses on the Android OS.

One of the primary malware propagation methods is through app markets. Figure 1.2

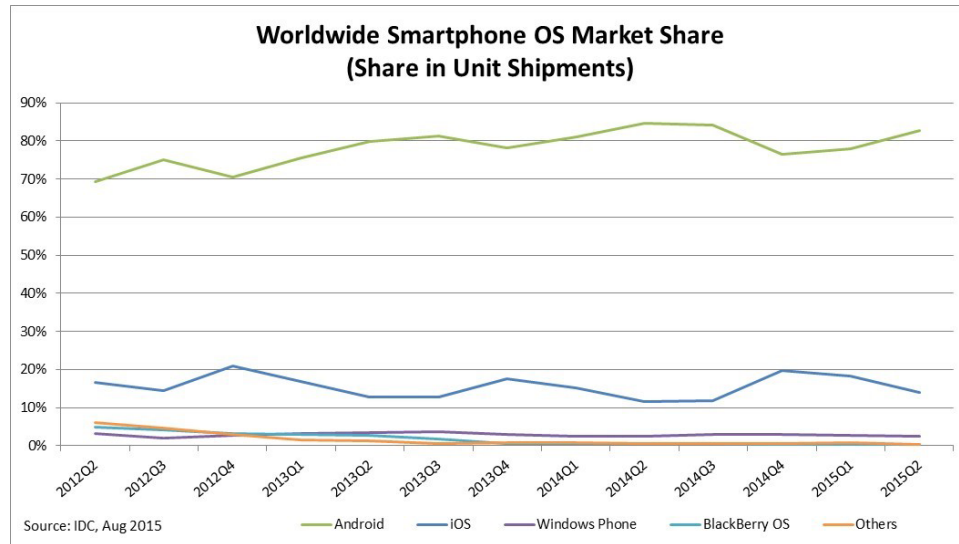


Figure 1.1: Smartphone OS Market Share(Android is the highest all the time)

shows the increase of applications in Google Play from December 2009 to February 2016 [56]. With this sheer increase in app numbers, the thorough analyses that might possibly reveal malicious activity would be infeasible.

There are two main strategies to analysis and classification of Android malware: static analysis and dynamic analysis. Static analysis methods focus on analysis of code without its execution. Typically, this means analysis of source code which is time and resource consuming.

Different from static analysis, dynamic analysis is more focused on the app during its execution time. Using dynamic analysis requires researchers to establish a simulation environment in order to monitor the behaviour exhibited by the running apps. One of the main challenges in this approach is to find the proper setting that deceives malware and triggers all malicious functionality. That is embedded in the code since ensuring that all possible behaviour was seen in problematic, dynamic approaches typically produces only partial results.

In summary, both static and dynamic analysis have great advantages; they focus on different aspects and have good accuracy but at the same time they are both time and resource consuming.

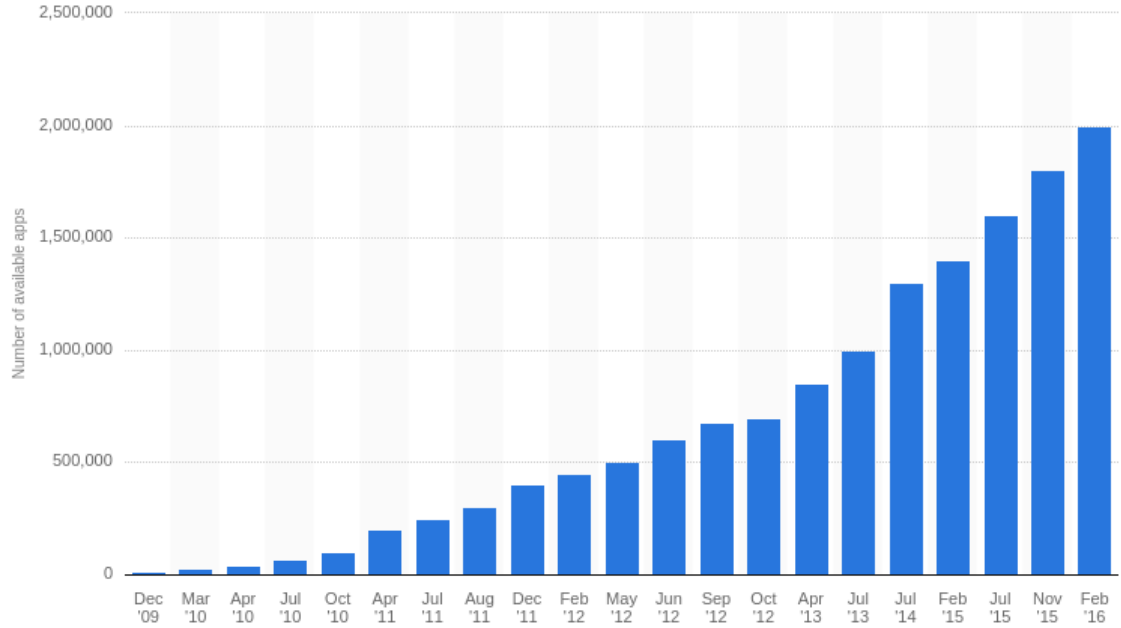


Figure 1.2: Number of available apps on Google Play market [56]

The large number of apps pushed to online markets every day makes manual analysis impractical. It is necessary for Android researchers to explore an approach which can filter the apps efficiently. As a result, anti-malware vendors apply lightweight triage techniques to quickly filter out known apps. “Triage is an area in which decision-makers must know what they are doing, why they are doing it, and which actions to take to achieve a satisfactory outcome” [40].

In our work, we offer an alternative approach that explores the role of meta-data in triage. For Android binaries, we focus specifically on meta-data information of each Android app without using any static or dynamic approaches. meta-data based analysis was previously attempted by a few studies that showed the potential of these features for malware classification [25][18]. Although promising, these studies are very limited in their conclusions and exploited only a few features. Yet, it is not clear why these features were used for analysis, which ones are ultimately suitable for filtering/classification of suspicious apps and why.

We focus on meta-data features extraction and their usage for Android apps triage.

Meta-data features are typically obfuscation resistant since code obfuscation is gaining popularity in the Android world. We propose meta-data based filtering of Android apps. Our contributions are as follows.

- We propose the approach to Android app triaging based on a combination of meta-data features including size, timestamp and etc. Within this study we explore over 300 meta-data features and assess their potential for an efficient and accurate triage of Android apps. To the best of our knowledge, this is the first attempt to systematically evaluate a large set of features for their applicability to Android apps detection.
- We further provide explainable detection. Our features allow us to understand why an app is found to be malicious or benign.
- To facilitate further research in this area, we present a framework for feature extraction and analysis.
- We conduct our experiment on a comprehensive set of over 150000 Android apps. To this end, we collect a diverse, comprehensive and recent dataset from 8 Android markets around the globe. Among the existing datasets, our set is the most representative of today's Android app environment.

Chapter 2

Related work

The amount of research on analysis of Android apps has been increasing in recent years. A broad overview of existing studies in this domain was conducted [25]. However, malware analysis can be split into two broad categories: static analysis and dynamic analysis. Static analysis focuses on analyzing dex files, extracting information from source code or bytecode in order to determine whether or not there are any suspicious points without executing the Android app. Dynamic analysis would set up a simulated system, install the unknown APK into a simulated environment then monitor the APK execution behaviour and record the running data which is used for the analysis of suspicious content.

In the following sections we introduce existing studies based on these two types of analysis.

2.1 Background on Android environment

“Each digital file is accompanied by meta-data information. meta-data is data that describes other data. Meta is a prefix that in most information technology uses means an underlying definition or description.” [49]. Meta-data can be automatically generated or created manually. Meta-data is important; it includes side information

for each file such as creation time, modified time or compressed/uncompressed size. In our work, our goal is to extract meta-data features from Android apps for our analysis.

Each Android app is packaged into an APK file [29]; the architecture of an APK file is shown in Figure 2.1.

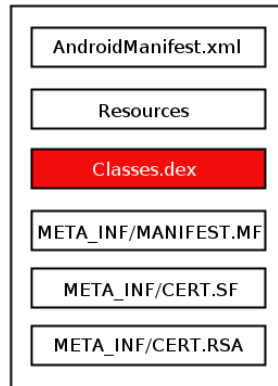


Figure 2.1: APK file format [29]

An APK contains a XML file which defines the structure of the source code, the AndroidManifest.xml. The XML file is a mandatory file for each Android app to have which contains the skeleton of the app and includes all elements required by the Android system. Elements from an AndroidManifest.xml file include activities, services, broadcast receivers and related contents. “An activity represents a single screen with a user interface just like window or frame of Java.Android activity” [61]. If there are ads or notifications included in activities, there must be corresponding elements included in the layout files because they must be visible to the users. From layout files, we can easily find out which ones have ads and what other visible content is included. “A service is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if the application is destroyed.” [2]. “Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometimes called events or intents. For example, applications can also initiate

broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who intercepts this communication and initiates appropriate action.” [1]. Unlike activities, both services and broadcast receivers are invisible to the users. Users are not directly aware of what is going on behind the scenes. One way to decipher the functionality of services and broadcast receivers is to analyze the source code of the app. This belongs to static analysis; based on this approach, researchers are able to analyze the background processes. The other efficient way is using dynamic analysis; with this approach, analyzers are able to monitor the behaviours of an app and then figure out what activities are happening in the background. These contents are used in our work; we did not analyze the code or monitor the activities, but only parsed their names and used them for lightweight triaging.

AndroidManifest.xml also contains a set of permissions required for the app. For the Android system, there is a “permission” mechanism that enforces restrictions on particular operations. For each operation from an Android app to be performed properly, necessary permissions should be granted by the system in order to make sure the operation is able to access the specific data. So permissions are important in each Android app. The granting work is done by mobile users when they are using the app, but the necessary permissions should be included in AndroidManifest.xml. In our work, we did not do much analysis on permissions, but only extracted the names of permissions and processed them with lightweight methods.

An APK also contains resource files that includes some external files, bitmaps, strings, media files or other files to run an app. Some third party libraries require extra resource files to make sure the external libraries can run without any errors or exceptions. An APK is typically signed with a proper signature before being submitted to online Android markets after development. Signature files include company names, issuers’ names, and key validation. All information about the APK author

should be provided within the signature file and they are checked before the APK is uploaded to online markets. In this work, we include features extracted from signature files. Contents from resource files include a large number of manually created files, some of them are not auto-generated so we did not include many features from the resource files.

Android apps are written in Java. In Java programming, programmers widely use classes. Each class belongs to a package; for example, Rectangle class in the “java.awt” package is “java.awt.Rectangle”. So at this point, the package name of each Java file is the content before the last period(“.”) delimiter. The package name of java.awt.Rectangle is java.awt. In our work, we include some features processed with this principle.

2.2 Static analysis

A lot of Android analyses are done via static analysis. Alisa [54] mentioned the procedure with some approaches to gather data and also the necessary steps of creating a model for malicious program defence. In CLAPP [34], some analyzers did static analysis using features extracted from loops in source code as the fundamental element. Loops are special in all programming languages and for security problems. For example, an infinite loop could drain a users memory without them being aware. For another case, an attacker could write a memory erase program and trigger the program inside a loop, deleting all of the users content. The work was done directly based on Dalvik bytecode, and it therefore does not rely on having access to the application source code. The authors extracted detailed information about the operations that influence and control such as the number of iterations of a loop, and the operations that constitute the loops’ body. All their work is achieved by combining several static analysis techniques, such as loop identification, backward data-flow

analysis on use-def chains, selective abstract interpretation, and code reach-ability analysis.

Several studies were based on signature detection; they also use static analysis as the fundamental approach [53][21]. Apposcopy [33] also includes signature based malware detectors. The tool uses static methods to extract information and check if the new app contains a sequence of instructions that is matched by a regular expression. Another purpose for using static analysis is to detect suspicious information flows of the app.

DroidSafe [35], an information flow analysis tool; was created to detect suspicious data flows via static analysis framework and it is also built to check data leakage. Schmidt et al. classified malicious apps via machine learning algorithm and the features he extracted are from ELF(Executable and Linkable) binaries [52].

RiskRanker [36] utilizes a variety of static analysis techniques to detect malware on Android devices. These techniques include program control flow graph evaluation and evaluation of byte code signatures. AnDarwin [23] is designed for the scalable detection of plagiarized applications. This is performed through the construction of a Program Dependency Graph (PDG) of the Android application. A semantic vector is then extracted from the PDG to represent the application, which can then be quickly compared with the semantic vectors of other applications.

The majority of existing studies based on static analysis require a significant amount of work to parse, extract and analyze the data. Although this time-consuming process typically provides accurate assessment of app functionality, it is often not efficient for large scale analysis. Parsing files and computing resources takes a significant amount of time so if there is a lightweight approach which means using less time and resources to filter out those “obviously can be classified” APKs, then we can apply this approach and decrease the number of Android apps waiting to be classified in our dataset. As a result in this thesis, we explore a lightweight approach capable of

large scale processing of Android apps.

2.3 Dynamic analysis

TaintDroid [26], an efficient, system-wide dynamic taint tracking and analysis system, is used to track multiple sources of sensitive data. It provides a program to monitor a third-party application usage of sensitive information, for instance, what information leaves a device and where the destination is. It assumes that third-party applications are not trusted, and the monitors observe and record how these applications access and operate personal data in real time.

Crowdroid [22] provides another behaviour-based detection framework. The researchers built an information server which is used to monitor all applications non-personal system calls from users. The remote server is used for parsing data, and creating a system called vector for each interaction from the users in their Android apps. Thus, a dataset of behaviour data is created for every application used.

DroidScope [66], is an emulation-based Android malware analysis engine that can be used to analyze Java and native components of Android apps. It exposes an event-based analysis interface with three sets of APIs that correspond to the four different abstraction levels of an Android Device: hardware, Linux and Dalvik. They monitor the malware activities at the API level and look into how a malicious app behaves inside the system.

ProfileDroid [63] is another framework in which the analysis layer is divided into four: static, user, OS and network. For each layer there are 2 components, monitoring and profiling. In Andlantis [20], analyzers present a highly scalable dynamic analysis framework. The framework runs the Android OS in a virtualized environment, it captures an application disk and network activity.

From the descriptions of dynamic analysis above, they all need virtual machines to

simulate the environment for different types of monitoring and the monitoring work would always take a long time. Using dynamic analysis would help researchers have a better understanding of APK behaviour. For some malware families like SMS or ransomware, dynamic analysis is practical because the researchers can find suspicious behaviours from activities such as network connection or message sending from those APKs. At the same time, researchers do not need to worry about obfuscation techniques used on source codes from the attackers, because they do not need to extract features from source codes or bytecodes in APKs. Meanwhile, dynamic analysis costs a lot, especially in terms of processing time; researchers need time and devices to build the environment also it is time consuming for upgrading and configurations of the OSs. Monitoring the large number of APKs brings significant overhead that adds to app analysis.

There are a few studies that used combinations of static and dynamic analysis. Zheng et al. provide a system which extracts features from op-code and manually generates signatures based on methods, names and whole applications [68]. After that, they compare the content and work on obfuscation detection. At the same time, the researchers also did some work to analyze malware with attachment files and dynamic payloads. From the analysis work [55], the analyzers created a sandbox which can be used for everyone through a web interface. In their work, they used a novel combination of static and dynamic techniques; meanwhile they track native API calls and extract features based on the APIs.

2.4 Metadata

The resource consuming nature of static and dynamic analysis based techniques was acknowledged by a few studies [18][32][48]. This turned our attention to lightweight meta-data features. meta-data information is side data that accompanies every APK.

Research study	Android meta-data features
Droidganger [67]	Permissions
PermissionWatcher [57]	Permissions combination of apps
Barrera et al. [17]	Permissions
Armando et al. [15]	Permissions
Kelley et al. [39]	Permissions
RefineDroid [38]	Suspicious permissions
Sato et al. [51]	Permission, Intent filter (action), Intent filter (category) and Process name
Wijesekera et al. [64]	Permission usage by third-party applications
Stowaway [30]	Permissions and system API calls
Achimescu et al. [14]	Image files
Teufl et al. [58]	Last time modified, category, price, description...
Munoz et al. [45]	Intrinsic application features, application category, developer-related features, certificate-related features, social-related features
MARVIN [41]	owner, validity, permissions, activities, receivers ...
DREBIN [16]	network addresses, permissions
DroidMat [65]	deployment of components, intent messages passing, permissions

Table 2.1: Existing studies that employ meta-data features in analysis

It includes information about app creation time, its size, permissions, resources and other data necessary for app execution. A brief summary of studies using meta-data features can be found in Table 2.1.

One of the most commonly used existing meta-data features is permissions. In essence, permissions describe the main tasks of an app. For example, in order to access the Internet, `android.permission.INTERNET` statement should be placed in the `AndroidManifest` file of an APK, another example being, the permission `android.Permission.READ_CONTACTS` is used by an application to access the contacts of a user. The permissions are enforced when the application is executed and the permissions not granted yield errors [48]. In an Android app, the author must declare all necessary permissions in the `AndroidManifest.xml` file in order to access

the resources in OS properly; this fact was leveraged by existing studies.

Kelley et al. [39] did an analysis focusing on users' understanding of permissions requested when they install the apps and their conclusion is "users do not understand Android permission and they are not currently well prepared to make informed privacy and security decisions around installing apps from Android markets".

Sarma et al. [50] analyzed the risk based on permissions on the Android platform. They proposed the notion of using effective signals to enhance Android security, introduced risk signals with the combination of the permissions requested by an app and the function category of an app as well as what permissions other apps request. Several studies analyzed the permissions commonly used by legitimate apps and malware [32]. A lot of users do not understand what each permission means and blindly grants them without any consideration, allowing the application to access sensitive information of the user. On top of that, malware apps often grant themselves necessary permissions without user consent.

Studies that focus on meta-data features often use permissions as the main and sometimes, only feature in analysis. Felt et al. [31] evaluated whether permissions of an Android app are effective at protecting users after collecting the permission requirements of a large set of Google Chrome extensions and Android applications. Based on their work, they concluded that installation security warnings may not be an effective malware prevention tool. PermissionWatcher [57] was introduced to help classify Android apps with probability risks to the users based on a set of rules and attack scenarios. The approach correlates resources to access permission combinations, and based on the permissions included in each Android app, they analyze possible attacks. Sato et al. [51] also looked at analysis of manifest files in Android apps by extracting four types of keyword lists: (1) Permission, (2) Intent filter (action), (3) Intent filter (category), and (4) Process name. This approach determines the malignancy score by classifying permissions individually as malicious

or benign.

Several studies analyzed contextual integrity for Android permissions. Wijesekera et al. [64] worked on quantifying the permission usage by third-party applications under realistic circumstances. Based on their work with users, they suggest that some requests should be granted by runtime consent dialogues rather than Androids previous all-or-nothing install-time approval approach.

Felt et al. [30] introduced Stowaway: a tool that uses both static analysis and permission features in order to detect overprivilege. They use static analysis to determine permissions used for API calls and compare them to the set of permissions requested. Permissions analysis based tools were extended in many other studies. Barrera et al. [17] presented an analysis including permission-based security models. They found only a small number of permissions include functionality on devices. In their work, they identified application clusters based on requested permissions and extracted the prominent permissions within each cluster. The most important part of their work was the creation of Self-Organizing Maps based on apps that represent the permissions. Armando et al. [15] presented an extension of the Android permission system and security framework which relied on existing resources and can be implemented on top of standard Android distributions. RefineDroid [38] looked for suspicious permission uses, e.g. access to unexpected Internet domains. Droidganger [67] provides information of permission usage to users using two techniques: record/replay and permission suppression. The researchers showed that their work can help users have a better understanding of about 40% of application permissions.

The majority of studies focusing on meta-data features also employ static analysis techniques as a hybrid technique for app analysis. Besides permissions, several other meta-data features were also explored. Achimescu et al. [14] worked on image analysis on Android. The researchers concluded that combination extracted from JPEG and EXIF files could provide a powerful authentication method.

In more extensive study by Teufl et al. [58], the researchers collected a list of meta-data from Google Play market. From Peters work, a set of features were analyzed for their potential of malware detection; this set included last time modified, category, price, description and some other features. His analysis are based on these features and involved some statistical analysis including clustering algorithms and natural language processing of the meta-data. Other researchers using features from Google Play in their work, Munoz et al. [45], extracted features from different categorizes of data including application category, developer-related features, certificate-related features, etc. These features are used for the analysis of security threats.

MARVIN [41], creates a malice score based on machine learning techniques to assess the risk, it is a system that combines static with dynamic analysis. In that work, the meta-data features selected by the researchers are mainly permissions. DREBIN [16] is a lightweight method for detection of Android malware automatically and directly on smartphones. The researchers extracted static analysis features including meta-data features. Mainly, they worked on strings, permissions and API calls to create feature vectors. In DroidMat [65], the researchers proposed a feature-based mechanism to provide a static analyst model for Android malware detection. In their work, they have features including permissions, API calls and their classification work is via kNN algorithm.

In previous work, most of the researchers extracted features and applied them for static and dynamic analysis. Features were extracted from all fields, from texts, side files, system calls and some other deep layers such as byte code and op-code. Meanwhile, only a limited number of studies have focused on meta-data and nobody has provided a clarification about how meta-data can be used for Android analysis. In this work, we provide a detailed analysis of using meta-data features and based on this analysis we propose an approach for malware triage.

Chapter 3

Design

In this work, we build a system framework including modules for extracting and processing features with feature selection and machine learning. Our main work is to create a framework for Android malware triaging using meta-data features. In order to do this, several objectives must be satisfied:

- Accuracy: To provide effective triage, the analysis framework has to be capable of identifying suspicious Android apps with high accuracy and low misclassification rates.
- Efficiency: The work is based on meta-data features solely without any features extracted from static or dynamic analysis. There is no simulator configuration nor source/byte code parsing; all the features are lightweight so the features extraction should be efficient.
- Flexibility: It should be simple to update with a flexible system. Currently we already applied our features but if there are any new features coming it should be easy to include them.
- Signature free: Since currently existing signature-based systems showed their unsuitability; our framework has to be capable of triage independent of man-

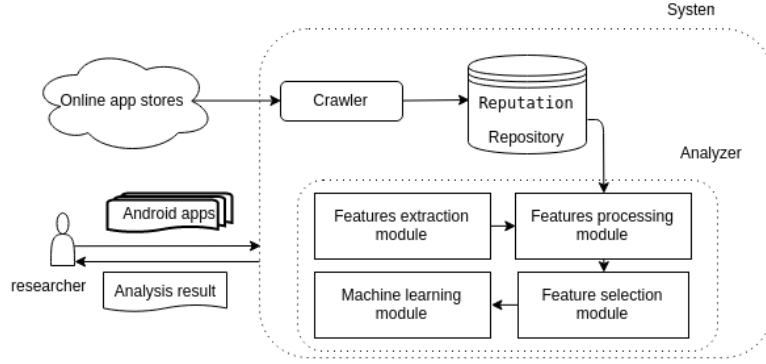


Figure 3.1: Overview of our design

ually created malware patterns.

The workflow diagram approach is shown in Figure 3.1.

The process of our framework is illustrated in Figure 3.1 and outlined in the following:

- Given an Android app, the meta-data raw features extraction module extracts all raw features from the app. We extract features alone from 6 major categories: size, timestamp, reputation of package names, authors, AndroidManifest files' elements and tools. We designed our parsers and extracted the necessary items from each app with our parsers. Different features are extracted using different parsers. The challenge for this module for us is to find an efficient approach for implementing the feature parsers; there are different ways of extracting the same features but we need to seek the most efficient one who takes the least amount of time and resources. Meanwhile, apps are created with different tools which makes them various. A feature parser may not work well on some of the apps because of their structure, so we need to be concerned about all exceptions we may have. Also in this module, meta-data features were extracted from all files in an app because they all have equal weight.
- After we collect the raw features from the app, the features are then transmitted to the feature processing module in our system, the meta-data raw features processing module. In this module, the goal is to use statistical approaches

with reputation generated previously to process and finalize the raw features. The system needs to refine and process them. Two goals are included in this module: deriving necessary statistics from raw features and representing these features in a format suitable for analysis. The challenges when we need to implement this module include:

- We need to find out how to process the raw features. Different features should be applied with different processing approaches in our system. We need to have proper parameter settings to apply with statistical analysis. Meanwhile reputations should be properly used for processing raw features. For example, in order to explore the potential of timestamps and size features deeper, we extracted different types of files, the files in the library folder, in the resources folder, images files, total files, also ratios between different values in order to extract as much information as we could. We do not have parameters for size features; just based on their file types and also the size of the same files for uncompressed and compressed states.
 - We need to find an efficient way to process the raw features with less time and resource overhead. Compared to extracting raw features, we still need to build an efficient mechanism for processing instead of extraction.
 - Android apps are diverse; the raw features are different from each other as well. We need to consider any exceptions we design to process raw features. The same processing work may not be suitable for features extracted from some abnormal apps.
- Once raw features are successfully refined and processed, the next step is to use them in triage. Before the system starts our machine learning classification, it includes a feature selection pre-processing step. In this step, the system would

pick out the proper features which are suitable for our classification. The challenge for us in implementing this module is choosing the feature selection algorithm.

- The last module of work in our system is the machine learning module. In this module, the system needs to apply the selected features from previous modules and return the result to the user.

In our system, the crawler crawls apps from different online app stores in order to build the dataset for our analysis. The analyzer has four different modules: features extraction, features processing, features selection and machine learning.

Apps crawled from online markets are used for creating reputations (benign rate which is explained in the next chapter). The reputation of the elements is used in our system for processing the raw features.

This analysis is based on analyzing the use of meta-data features for malware triage so we did not plan to spend much time on machine learning and accuracy increasing. However, we still need to demonstrate our contribution so we need to find a machine learning algorithm that could provide a promising accuracy based on our features. In the following chapter, we discuss the features, processing work and parameter settings in more detail of our system framework.

Chapter 4

Features

One of the fundamental problems of mobile app triage is the selection of features that provide accurate detection while remaining scalable for automatic extraction and analysis. Traditionally, the mobile security studies focused on complex sets of characteristics extracted through static and dynamic analysis of an executable files content. While these methods offer good accuracy, the complexity related to these characteristics extraction and analysis is often prohibitively expensive for large scale triage of Android apps.

In this work, we turn our attention to an alternative approach and focus on meta-data associated with Android apps. Our approach is inspired by a recent study conducted by Teuffl et al. [59] where authors attempted to use meta-data content to classify Android apps. The set of features used in this study primarily focused on attributes related to apps location on the market (in this case, the study looked at Google Play market). The features deemed useful by authors are given in Figure 4.1. Out of these features, only the first seven were employed in this study and many other features only exist in apps found on the Google Play app store. For example, the price tag could be different for the same app in different markets. So in our work, we focused on the features extracted from APKs themselves rather than from

Feature	Description
Last time modified	Last time the application was replaced with a new version
Category	Application category (e.g., games, travel, and reference)
Price	Numerical and textual representation of application price
Description	Textual description of application
Permissions	Permissions required by the application
Rating	Average rating of application
Number of downloads	Ordinal value describing the number of downloads
Creator ID	Short user name chosen by the developer
Contact email	Contact data of the developer
Contact website	Website of the developer
Promotional video	Link to a promotional video describing the application
Number of screenshots	Number of screenshots used within the application description
Promo text	Short promotional text for the application
Recent changes	Description, which is related to recent application changes
ID	Unique application ID (changes with new version)
Package name	Application package name (e.g., org.example.androidapp)
Installation size	Installation size of application (in bytes)
Version	Textual and numeric description of application version code
Application type	Application type (e.g., library and application)
Ratings Count	Number of given ratings
Application title	Title of application

Figure 4.1: A list of feature employed in the study by Teuff et al. [59].

features related to app stores.

In our thesis, we experimented with an extensive list of over 300 meta-data features. We grouped these features by major categories. These categories and their brief descriptions are provided in Table 4.1. Each of these categories are explained in greater detail in the following sections.

4.1 Size

Size is one of the fundamental meta-data features that exists for every file in a computer. In Android app analysis, size plays an important role. Based on our

Feature category	Description
Size	Features related with the size of files and the feature group composed with the size of APK files, such as size of dex file, size of image files, etc.
Timestamp	Timestamp features could provide information about when and how Android APK is created. These features are also used to describe the number of files created during different time intervals.
AndroidManifest file analysis	Activities, services, permissions and some basic elements are listed in the AndroidManifest files and there are many strings in the manifest files, including names, etc. based on those string elements in the manifest file
Reputation analysis	This group includes signature analysis and package name analysis
Certification and signature analysis	For each signature, there is information such as issuer, owner and their MD5 signature. For some APKs, more than one MD5 signatures are included.
Tool analysis	Based on some patterns for the APKs made by different tools, such as Android studio, Adobe air. So this feature is based on the tools which are used for making apps

Table 4.1: meta-data feature groups

preliminary analysis of malware Android apps, we noticed a number of discrepancies that made an APK file questionable. Overall, the size feature group included 44 features. The list of the features given in Appendix Table A.1.

In our work, the size feature group includes features of different types of files and their percentage of the total size of an APK. There are plenty of resources online that allow attackers to create Android apps by simply injecting segments of malicious code into the benign apps and packing them back to their original form, (This is known as app repackaging). The intuition behind this is straightforward; if apps have identical size related features they are likely to share content. It could also be used as a lightweight approach for detecting APK repackaging.

dexafterzip	The number of dex files created after the apk file packaging time. In other words, the timestamp of dex files is greater than the APK creation time. It is possible for an APK to have more than one dex file.
dex2mins	The number of dex files created 2 minutes before the APK file creation time.
dex2hours	The number of dex files created between 2 minutes to 2 hours before the APK file was created.
dex1day	The number of dex files created between 2 hours to 1 day before the creation time of the APK file.
dex1month	The number of dex files created between 1 day to 1 month before the creation time of the APK file.
dexlonger	The number of dex files created more than 1 month before the APK file.

Table 4.2: Dex file features list

4.2 Time-stamp

Another group of meta-data features is related to time-stamp. Each file has a time-stamp label that represents the creation time or when it was last modified. Time-stamps can be used to help determine APK developing tools because each APK tool has a unique manner of creating an app and timestamp that can be used to discover that manner. For example, using Android Studio to create APK automatically results in a 2 minute gap between some files.

The following Table 4.2 provides descriptions of the features included in our time-stamp feature group.

There are 4 intervals used to further analyze time-related features: 2 minutes, 2 hours, 1 day and 1 month. These intervals were chosen based on our preliminary analysis (more details are provided in Chapter 6). The features produced for each of the files are given in Table A.2 in Appendix.

In addition to our raw features extracted directly from APK files, we also incorporated some statistics that give a better insight into the APK creation timeline in our analysis. We put in some features calculating the percentage ratio of files from

different time periods; in the following Table A.5 in AppendixA we give descriptions about ratios for files in the assets folder. Not only do we calculate the number of files that belong to each time interval, but also the ratios are calculated because they provide some overall knowledge for Android apps. Based on the ratios, we can get a better understanding of the apps developing process. For instance, based on our preliminary analysis, if most of the files from an APK belong to a time interval but the dex file and signature files belong to another time interval then this APK has a higher probability to be a repackaged app.

The features produced based on ratios are given in Table A.3 in Appendix.

In the Android developing process, signing an APK is always the last necessary step (aligning is not compulsory). During the last step, a META-INF folder is created with three files inside, CERT.RSA, CERT.SF and MANIFEST.MF. Typically, all these files should be created before the final APK and therefore their timestamp should be prior to APK packaging time. Our preliminary research showed that this is not always the case; for example, in certain situations, there are some APKs in our dataset including files with timestamp after APK packaging time which is suspicious. In our preliminary research, using some tools to repackage an app, inserting some third party files or libraries affected the time line of an app. Based on that, we considered that there must be some unjustifiable work from the APK makers, but at this point we cannot determine whether the APKs that meet this condition are malicious. To indicate the time line of the file creation we use the features shown in Table 4.3.

Overall timestamp feature group Table A.2 in AppendixA contains 147 features related to the timestamp of the files in an APK. Timestamps analysis could help detect Android development tools in other ways. Based on our research, the most famous tool used for decompiling Android apps is “APKTOOL”. After recompiling apps, developers need to sign the apps with their certifications using some other

afterMANIFEST_MF	The number of files with greater timestamps compare to MANIFEST_MF
afterCERT_RSA	The number of files with greater timestamps compare to the CERT_RSA file
afterCERT_SF	The number of files with greater timestamps compare to the CERT_SF file
afterMANIFEST_MF/totalfile	The ratio of afterMANIFEST_MF to total number of files
afterCERT_RSA/totalfile	The ratio of afterCERT_RSA to total number of files
afterCERT_SF/totalfile	The ratio of afterCERT_SF to the total.

Table 4.3: Timestamp features related to signature files

tools. Repackaging apps is a classic way for creating malware, so in our work, we included tools analysis and tried to determine if specific tools are prone to creating malware. Each tool has a unique timestamp pattern when it is used for assembling APKs; for instance, some APKs would have a static number of time gaps between the signature file and the dex file, some tools set the same timestamp for all files in the same folder and some tools automatically generate files with a particular timestamp and all of these features could be used for different analyses in future work.

4.3 Reputation

In our work, reputation plays an important role. We use reputation for processing different raw features including package names, authors and elements from Android-Manifest files. Reputation is used to express the legitimate degree of elements in our work. The reputation of each element reflects the nature of the apps including it. The reputation of an activity is the benign ratio of the activity and it is calculated by the number of legitimate apps including this activity to the number of all apps

tag	android_name	benign_rate	BenignOrNot
activity	com.dgame.studio.dont.pileup	0.142857142857	malicious
activity	com.nduvpyn.asidbro148530	0.0	malicious
activity	com.ggbook.recharge	1.0	benign
activity	com.qidian.QDReader.ui	0.0	malicious
activity	LWPServiceWelcome	0.0769230769231	malicious
activity	com.scringo.features.inbox	0.894736842105	benign
activity	com.education.DooFree	1.0	benign
activity	com.helpshift	0.992	benign
activity	com.quickmobile.conference.mynotes	1.0	benign
activity	com.doapps.android.mln.debug	1.0	benign
activity	com.dowscape.near.app.activity	0.8	benign
activity	cordovaExample	0.931818181818	benign
activity	UE3JavaApp	1.0	benign
activity	com.chartboost	0.4	malicious
activity	com.gbulwhxitt.hbnprugtsl103926	0.0	malicious
activity	info.grupovg.websplashrate	1.0	benign
activity	com.kugou.fanxing.user.activity	0.571428571429	benign
activity	tvya.kyr	0.0	malicious
activity	com.yong.gift	0.833333333333	benign
activity	ArchivNotification	1.0	benign
activity	cn.iyd.slideview.activity	0.0	malicious
activity	com.gigya.socialize.android	1.0	benign
activity	com.unity3d.player.z	0.571428571429	benign
activity	com.freshplanet.googleplaygames	0.923076923077	benign
activity	com.soomla.profile.social.facebook	1.0	benign
activity	com.ziplinegames.moai	0.590909090909	benign

Figure 4.2: Sample activity database from analysis

including the activity. A reputation is a lightweight feature to represent a benign or malicious tendency of an item.

The reputation system based on each element tag and a sample reputation content table for activities is shown in Figure 4.2: the first column is the name of the proper tag, the second column is the processed name of each activity, the third column which is also the most important one, stands for the benign ratio of each activity and the last column is the overall status of the activity. The benign ratios (reputation) are in the third column.

Features used for reputation calculation: package name, author and elements from AndroidManifest file. The reputation for AndroidManifest content is used for the features in AndroidManifest analysis.

In this section, there are two features directly based on reputation and used in our work. The name of the features and descriptions are in Table 4.4. For the package names of apps, we remove the content after the last delimiter. The content before

Reputation features	Description
benignNum	The number of legitimate apps with the same author of the current Android app
ESET	The number of malicious apps with the same author of the current Android app
total	The number of apps with the same author of the current Android app
repRatio	The reputation of the app creator
pbenignNum	The number of legit apps with the same package name as the current Android app
pESET	The number of malicious apps with the same package name as the current Android app
ptotal	The number of apps with the same package name as the current Android app
prepRatio	The reputation of the app package name

Table 4.4: The list of features based on package name and author reputation

the last delimiter always stands for a company representative; we are focusing on reputations for a cluster not a single person. For example, the package name of an app is “com.example.art” and in our work, the package name is “com.example”.

4.4 Third-party libraries and AndroidManifest file analysis

Every application must have an AndroidManifest.xml file in its root directory. The AndroidManifest file presents essential information about an app to the Android system. An example of AndroidManifest.xml is given in Figure 4.3.

Compared to previous meta-data features, features extracted from the manifest file have special purposes and status. While features in the size and timestamp groups relate to files within an APK, the manifest file contains the basic skeleton of an APK and every element in this file is closely related to content of the source code.

In AndroidManifest file, each element has a specific purpose and functionality. For our analysis, we use name strings from nine elements: **action**, **activity**, **meta-data**,

```

- <manifest android:versionCode="2" android:versionName="1.1" package="aero.developer.androidsample.app">
  <uses-sdk android:minSdkVersion="11" android:targetSdkVersion="19"> </uses-sdk>
  <uses-permission android:name="android.permission.INTERNET"> </uses-permission>
  <uses-permission android:name="android.permission.GET_ACCOUNTS"> </uses-permission>
  <uses-permission android:name="android.permission.WAKE_LOCK"> </uses-permission>
  <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"> </uses-permission>
- <application android:theme="@7F0B0000" android:label="@7F0A000E" android:icon="@7F020074" android:allowBackup="true">
  <meta-data android:name="com.google.android.gms.version" android:value="@7F090001"> </meta-data>
- <activity android:label="@7F0A000E" android:name="aero.developer.androidsample.app.MainActivity">
  - <intent-filter>
    <action android:name="android.intent.action.MAIN"> </action>
    <category android:name="android.intent.category.LAUNCHER"> </category>
  </intent-filter>
  <activity>
  <activity android:label="Boarding Pass" android:name="aero.developer.androidsample.app.BoardingPassDetailScreenActivity"
  android:stateNotNeeded="true" android:screenOrientation="5" android:noHistory="true"
  android:parentActivityName="aero.developer.androidsample.app.MainActivity"> </activity>
- <receiver android:name="aero.developer.androidsample.app.gcm.GcmBroadcastReceiver" android:permission="com.google.android.c2dm.permission.SEND">
  - <intent-filter>
    <action android:name="com.google.android.c2dm.intent.RECEIVE"> </action>
    <category android:name="aero.developer.androidsample.app"> </category>
  </intent-filter>
  </receiver>
  <service android:name="aero.developer.androidsample.app.gcm.LocalService"> </service>
</application>
</manifest>

```

Figure 4.3: An example of legitimate elements in AndroidManifest file

permission, provider, receiver, service, uses-feature, uses-permission. There are a few others (such as `uses-sdk`, `supports-screens` and etc.) that are less common and not always present in apps. The descriptions and usages of the nine elements are explained in the following list.

- **action:** It contains the name of the action, because action tag is always encapsulated in the intent-filter tag, therefore the name of action could be a class name. For example “com.example.activity2” which means the intent is used between two developer made activities or the name could also be an Android action; for example, all APKs should have a start layout and the action of that activity is “android.intent.action.MAIN”. For some other system actions there are some other names such as “android.intent.action.CALL”.
- **activity:** It always stands for the name of java classes behind the correlated layout, if the author wants to use activities from third-party packages, then they must list them in the manifest file. Simply based on the name of activities we can easily identify if the activity is from the APK package or An external third library package; this can be done by parsing the activity name by the delimiter “.” or with the package name. For example if the package name of an APK is “com.example.test” and one of the activities has an activity name

“com.example.test.activity1” or “.activity1” then it means this activity is from the developer made APK package not from external third party packages.

- **meta-data:** They stand for meta-data elements which are used for supporting an APK. For example in the Google ads Figure 4.4, the meta-data in the AndroidManifest file includes the version code of the proper Google API.
- **permission:** These are permissions defined by authors and they can be referred to as existing permissions after being defined. Most of the time, permission elements are developer defined permissions, or uses-permissions which are defined by Android which gives access to specific pieces of data. For instance, “com.tencent.ms.permission.account.syn” is a permission from tencent company just based on the name.
- **receiver, service and provider:** They stand for the name of the background processes behind the scenes because none of the three elements is visible to Android users. They are the processes which are hard for users to visualize and be aware also there might be some high risk behaviours from these elements. From our reputation data set, we extracted all services, receivers and providers from the APKs and we found there are tons of services from third party libraries; they are complicated and it is impossible to judge if they are benign or malicious simply based on their names.
- **uses-feature:** They specify a single hardware or software feature used by the application. In [7] and [12], there are two lists for hardware features and software features.

For the name attributes in **uses-permission**. They are the names of the system defined permissions. For each item, it could be a permission defined by the application with the permission element, a permission defined by an-

other application, or one of the standard system permissions (such as “android.permission.CAMERA” or “android.permission.READ_CONTACTS”)

In an AndroidManifest file, each element accompanied by attributes. All attributes are optional. However, at least some have to be present to accomplish its purpose. One of these attributes is name. For example, any advertisements included in an Android app has to be specified in AndroidManifest file. Under Android:activity, each such activity is named under “android:name” attribute. For another example, if an Android developer wants to attach Google advertisements, then he/she has to add the following content 4.4 into the manifest file.

```
<!-- Include required permissions for Google Mobile Ads to run-->
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <!--This meta-data tag is required to use Google Play Services.-->
    <meta-data android:name="com.google.android.gms.version"
        android:value="@integer/google_play_services_version" />
    <activity
        android:name=".MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <!--Include the AdActivity configChanges and theme. -->
    <activity android:name="com.google.android.gms.ads.AdActivity"
        android:configChanges="keyboard|keyboardHidden|orientation|screenLayout|uiMode|screenSize|smallestScreenSize"
        android:theme="@android:style/Theme.Translucent" />
</application>
```

Figure 4.4: An example of Google ads elements in AndroidManifest file

AndroidManifest file is important for each APK; for any advertisement to be included in an app, an activity with corresponding name attribute has to appear in the AndroidManifest file. For example, if an author wants to add advertisements from Google, then he needs to add meta-data data elements including the

version code which is a necessity for Google ads and the name of the content is “com.google.android.gms.version”. Besides this, he also has to add an activity that includes a name attribute having a value “com.google.android.gms.ads.AdActivity” which is a fragment used to lay out the ads to be visualized by users. Also the APK needs to connect to the Internet so two compulsory permissions, “android.permission.INTERNET” and “android.permission.ACCESS_NETWORK_STATE”, are listed in the AndroidManifest file in order to call network APIs because they need to connect to the Internet from the system. Those elements are necessary in the manifest file if the author wants to attach Google ads. As a result, the AndroidManifest file gives an overview of the content of the APK and sheds light on whether the manifest file contains unusual content. For example, based on our preliminary analysis the elements of the manifest file that contain unusual names (for example “com.zxsdfss.erqwa”) always indicate the presence of suspicious content from untrusted sources.

In this feature group, we extract all name attributes from nine given elements from all APKs. AndroidManifest file features include raw features extracted directly from apps and count based or statistical features. We include a list of features in Table 4.5.

actionNum	The number of actions in manifest file.
activityNum	The number of activities in manifest file.
providerNum	The number of providers in manifest file.
receiverNum	The number of receivers in manifest file.
serviceNum	The number of services in manifest file.
meta_dataNum	The number of meta-datas in manifest file.
permissionNum	The number of permissions in manifest file.
uses_featureNum	The number of uses-features in manifest file.
uses_permissionNum	The number of uses-permissions in manifest file.
underPackage	Counting the number of the elements from the current APK package, not from any external resources.

Table 4.5: AndroidManifest elements counting features

If the extracted name has a reputation we then use it to create some other statistical features.

Another attribute of features extracted from the manifest file is related to priority value. Priorities are used for managing the order in which the broadcasts are received, or used to force Android to prefer one activity over others. Normally, the priority value should be located within the range -1000 to 1000[10]; however there are some APKs with higher or lower priority values. Some Android makers set the maximum integer value to the priority in their apps because they want their service or broadcast receiver to have the highest priority compared to other apps. For example in Figure 4.5, which is extracted from an APK in our dataset, the priority is extremely high.

```
-<receiver android:name="com.fw.ttze.receiver.FwBReceiver">
  -<intent-filter android:priority="2147483647">
    <action android:name="android.intent.action.PACKAGE_ADDED"> </action>
    <action android:name="android.intent.action.PACKAGE_REMOVED"> </action>
    <data android:scheme="package"> </data>
  </intent-filter>
</receiver>
```

Figure 4.5: An example of AndroidManifest.xml file showing abnormally high priority value

Features related to priority in our work are in Table 4.6.

One of the features widely used in mobile security studies is a set of permissions. Permissions enforce restrictions on the specific operations of the Android system.

A list of default permissions was defined by [9], there is a list of manifest permissions; these permissions exist for all apps and do not need to be specifically added by developers. Based on the two lists above, we have a list of features related to them in Table 4.7.

A list of dangerous permissions are shown in [5]. In this work, we follow this list to calculate different categories of dangerous permissions.

As a conclusion, all features in this group are aiming to analyze the manifest files of

priorityNormal	The number of priorities with normal values which is within range -1000 to 1000
priorityAbnormal	The number of priorities with abnormal values which is lower than -1000 or higher than 1000
priorityAll	The total number of priorities in the AndroidManifest file
priorityNormalRatio	The ratio of the number of normal priorities to the total number of priorities
priorityAbnormalRatio	The ratio of the number of abnormal priorities to the total number of priorities
highestPriority	The highest priority value
lowestPriority	The lowest priority value

Table 4.6: Priority features

outAction	The number of external actions in manifest file in other words the number of developers defined actions from APKs in experiment data set.
outPermission	The number of external permissions in manifest file in other words the number of developers defined permissions from APKs in experiment data set.
outActionRate	The ratio of outAction to the total action number.
outPermissionRate	The ratio of outPermission to the total permission number.

Table 4.7: Permission features

APKs and try to find out how the use of external libraries of resources can impact the nature of APKs. The reputation of the elements in the AndroidManifest file would provide great help for triaging. Based on the reputations, it is relatively easy for us to have a basic understanding about the Android app. If an app includes low reputation elements, then it is likely to have more malicious content although we cannot directly label the app a malicious one. Compared with other meta-data features, information from AndroidManifest has a close relationship with the Android app so the features extracted from the AndroidManifest file are important. Features based on AndroidManifest analysis are all listed in Table A.6.

4.5 Certification analysis

Each Android app has to be signed by a developer. This signature is stored in the RSA file. The file contains information about the issuer and the owner as shown in Figure 4.6. We check the completeness of the information encapsulated in the signature.

```
Owner: CN=Thomas Visic, O=DataSoft, L=Scottsdale, ST=Arizona, C=US  
Issuer: CN=Thomas Visic, O=DataSoft, L=Scottsdale, ST=Arizona, C=US
```

Figure 4.6: Information about issuer and owner

Typically, the issuer of the APK has the same content as the owner. However, if the developer use some third-party proxies to help them signing apps then there maybe differences between the owner and issuer. We added a feature `sameOrNot` that stores a boolean value to identify if the issuer content is the same as author content. At the time the signatures created, usually the author needs to provide necessary details, for instance when creating a signature in Android Studio as in Figure 4.7

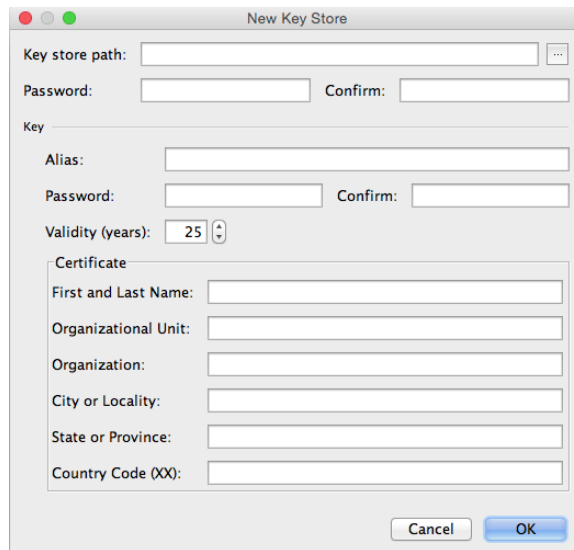


Figure 4.7: Create a keystore to sign APKs

In the certification box, a developer should fill his name, organization and all other information about his company. Some indolent developers just fill part of them and

keep the rest empty. We have a meta-data feature here used to store the number of items in the keystore **numOfItems** which identifies how many items have been filled in the certificate box in order to detect the completeness of the key store.

4.6 Tool analysis

The last feature in this section relates to analysis of Android developing/signing tools. There are multiple tools currently which can be used for making Android apps. However, as research showed certain tools are typically involved in development of app with malicious content. We employ meta-data features to explore this result. We have one lightweight feature called “toolname” which indicates the name of the development tool for an APK. An app built/signed by different tools has diverse structures and “toolname” feature is extracted based on the analysis on the structure. There are multiple tools for Android app development. In our work, we pick the following ones: Adobe Air, Keytool, JarSigner, Dot42, Eclipse and AndroidStudio. Some of the tools are famous for developing apps such as AndroidStudio and Eclipse, some of them are common for apps signing for instance: keytool and jarsigner. There are also some other tools but unfortunately we do not have large number of apps built from them. From our research, we noticed that there are some apps do not follow the pattern from any of the tools above so we added some other labels and details of this is explained in chapter 6.

In our research, we have identified some lightweight features for tools detection based on meta-data; they are not perfect but valuable. The number of APKs from each different tool is limited.

Chapter 5

Dataset

One of the common problems in the security domain is the lack of up-to-date data for experimentation. Several standard benchmark datasets produced by a few recent studies are quickly becoming obsolete. For example, The Android Malware Genome project data [69] and Drebin set [16] are limited in size and only include older versions of Android apps. Thus to ensure validity of our results, we constructed our own dataset. For our dataset we crawled eight markets from several different countries and collected over 190 000 apps during a period of 2 months from July to September in 2015.

We crawled and downloaded more than 53000 apps from six Chinese markets: Xiaomi, qihoo360, anruan, anzhi, gfan and tencent. According to [46], these are among the top 10 markets in China.

Another third party market we crawled was called Mobo market; it is from Hong Kong. We downloaded more than 92000 apps from this market and the reason we downloaded so many is that the Android apps in this market are made by Android developers from all over the world. In the other six Chinese markets, more than 95% of the apps are made by Chinese developers; however the APK developers' distribution is too narrow.

The last Android app store we crawled is the official Google Play market. It includes the largest number of apps among all online stores. We downloaded more than 45000 apps from Google Play for the analysis. Overall, we collected over 190000 apps during a period of 2 months in 2015. The resulting set was processed to remove duplicates based on their MD5 hash-values. To determine the number of legitimate apps, we used the Virustotal platform [62] to label the dataset. VirusTotal is an online platform that offers analysis of suspicious content. It includes more than 50 antivirus engines and website scanners. There are a number of online scanning platforms; Virustotal is a benchmark analyzer that is used by many research studies. Compared with other online analyzers, Virustotal platform runs faster, includes more samples and provides effective APIs for researchers to scan a large number of apps. Since there are over 50 different anti-malware scanners available on this platform, we chose to employ results of ESET-NOD32 [47] as it was reported among the best 10 antivirus applications in 2016 [60].

After we completed the duplication removal and scanning process, our final dataset included 159344 apps; the distribution of apps is shown in Table 5.1.

Market name	Number of Android apps	Benign apps	Malicious apps
Google play	40340	98.4% (39695)	1.6% (645)
Mobo market	80681	82% (66158)	18% (14523)
anzhi	11094	70% (7755)	30% (3339)
anruan	11091	72% (7986)	28% (3105)
gfan	3026	70% (2118)	30% (908)
tencent	7060	73% (5154)	27% (1906)
xiaomi	5043	68% (3429)	32% (1614)
360 market	1009	73% (737)	27% (272)
Total	159344	83.5% (133032)	16.5% (26312)

Table 5.1: Dataset app distribution

As Table 5.1 shows Google Play has the highest (98.4%) benign apps proportion compared to other markets, Mobo market has 82% of benign apps and Chinese markets have 70.4% benign apps. Based on the results scanned by ESET, the majority

of malware apps from Chinese markets were labelled as adware.

5.1 Dataset for reputation establishment

One of the components we employed in our approach is reputation analysis. To calculate reputation we used 119436 apps from our collected data. In this analysis, we have two features (package reputation and developer reputation). We needed to build reputations for package names and developers based on the information from some APKs. In order to do this, we used 3/4 of this dataset for information extraction and reputation calculation, and the remaining 1/4 for our experiment. Table 5.2 shows the splitting of our dataset into reputation dataset and experiment dataset.

Dataset Name	Number of apps	Benign apps	Malicious apps
Reputation dataset	119436	99278	20158
Google Play for reputation dataset	30237	29753	484
Mobo market for reputation dataset	60474	49589	10885
Chinese market for reputation dataset	28725	19936	8789
Experiment dataset	39908	33054	6854
Google Play for experiment dataset	9977	9817	160
Mobo market for experiment dataset	19954	16362	3592
Chinese market for experiment dataset	9977	6875	3102

Table 5.2: Details of app distribution in reputation dataset and experiment dataset

In the rest of this work, we use **reputation dataset** to stand for the 119436 apps in reputation systems and **experiment dataset** to stand for the 39908 apps for experiment. Based on information extracted from the reputation dataset, we built a reputation system based on three different reputation sub-systems: one is the developer reputation sub-system, another one is the package name sub-system and the last sub-system is based on elements extracted from AndroidManifest files from apps. In the developer reputation sub-system, we extracted MD5 hash values of app

developers using *keytool* command in Linux. The output from *keytool* of a sample APK is shown in Figure 5.1.

```

cn.zhui.client1753172_80647200.apk
Signer #1:

Signature:

Owner: CN=Zhui.CN, OU=Byban Ltd., O=Byban Ltd., L=Shanghai, ST=Shanghai, C=CN
Issuer: CN=Zhui.CN, OU=Byban Ltd., O=Byban Ltd., L=Shanghai, ST=Shanghai, C=CN
Serial number: 4b8a6a19
Valid from: Sun Feb 28 09:05:29 AST 2010 until: Fri Apr 18 10:05:29 ADT 2092
Certificate fingerprints:
    MD5:  00:46:9D:70:1B:96:B9:FC:D9:F6:AA:4C:1A:80:44:FB
        SHA1: 14:F4:04:13:4B:21:62:7B:3B:7F:F4:FE:8B:1D:C5:79:7F:51:EC:4F
        SHA256: 62:89:EB:22:09:90:9B:17:4A:29:F5:65:FF:F8:5B:1D:50:0D:9F:27:F8:2A:4E:4A:52:F5:FE:90:C1:78:FD:3B
Signature algorithm name: SHA1withRSA
Version: 3

package: name='cn.zhui.client1753172' versionCode='89' versionName='2.8.2.12' platformBuildVersionName=''

```

Figure 5.1: Sample APK signature

We extracted MD5 hash values from the output and package names to make developer¹ and package reputation sub-systems. The reputation developer sub-system is represented by app information extracted from developers present in our dataset. It includes the “**number of total apps**” representing the number of apps from the current developer, “**number of benign apps**” representing the number of benign apps from the current developer, “**number of malicious apps**” representing the number of malicious apps from the current developer and the “**benign ratio**” of the current developer which is calculated by the formula which indicates reputation score:

$$\text{benign ratio} = \frac{\text{number of benign app}}{\text{number of total app}}$$

For each developer in our dataset, we calculate the reputation score, which indicates the likelihood of an app produced by this developer being malicious.

This score was only calculated for developers with more than 5 apps in our dataset. Figure 5.2 shows the majority of developers in our dataset have at least 5 apps.

Another reputation sub-system is based on the package name, the “reputation package name sub-system”. The only differences are the next two points. In the developer reputation sub-system, we set a threshold of 5 and the developers who have more

¹Developer of an app is identified by an app signature

than 5 apps created could have reputations but in the package name reputation system, we just recorded all of the package names and their related information. We chose 5 based on the result from Figure 5.2. It shows the developers with the number of the apps created by them. We aim to increase the accuracy meanwhile trying to include more developer information so we consider 5 more or less is the value we need.

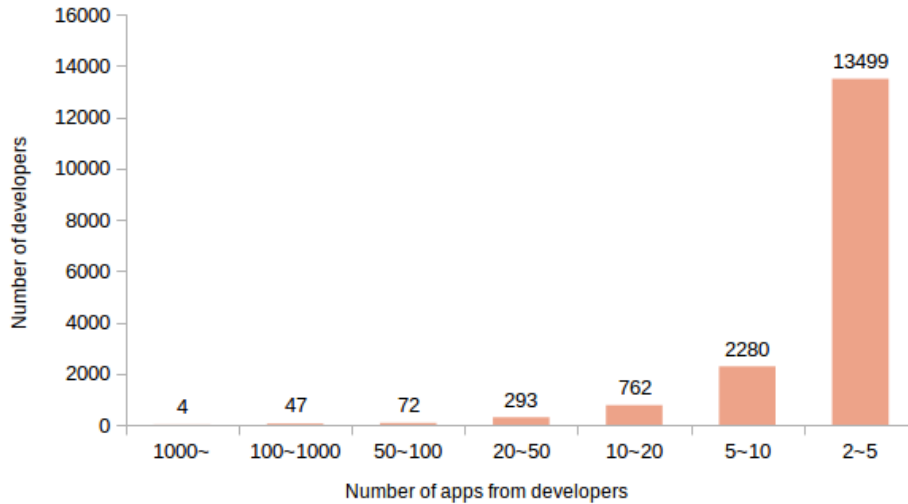


Figure 5.2: Distribution of apps by developers

One more change compared to the developer reputation sub-system is the name modification. In the developer reputation sub-system we just used the full MD5 hash values to stand for the developers, but in the package name reputation sub-system, we modified the package name. We removed the content after the last delimiter in the package name; for example, “com.youtube.scan” is converted to “com.youtube”, “com.huawei.cn.media” is converted to “com.huawei.cn”. The reason for doing this is based on the behaviour of Android development tools: generally, content behind the last delimiter is just a class name in Java; the content before is the package name. For example, there are four class names here: “a2z.Mobile.Event1997” “a2z.Mobile.Event2473” “a2z.Mobile.Event3072” “a2z.Mobile.Event3091”; all four APKs might share similar content or similar third party libraries or packages. Re-

moving the last content in a package name is a lightweight aggregation to categorize different APKs into one class so after this step all of the four APKs have the same “new package name”: “a2z.Mobile”. The last reputation sub-system calculates reputation for elements in AndroidManifest files and the details of this are explained in the next chapter.

Chapter 6

Experiment setup

In previous chapters, we talked about the dataset we have, the features we included and the design of our experiment. In this chapter, we explain the details of our implementation.

6.1 Implementation process

In our work, we built a framework containing four main parts: apps crawling, dataset building, reputation system creating, feature extraction with statistical processing and feature selection applied with machine learning classification.

Crawlers. To collect necessary data, we implemented several crawlers. The source code for crawlers was borrowed from several sources [46] [24] and customized for our own needs using Python and the Scrapy platform.

Our framework was developed using Python version 2.7 with the following libraries: subprocess, zipfile, datetime and xml.etree.ElementTree. These library utilities were necessary for processing management, ZIP file information extraction, timestamp information extraction and information extraction from XML files.

For developers' information and package name extraction, we used Linux command "keytool" including necessary parameters; the full command is "keytool -list

-printcert -jarfile” to extract the signatures of the apps. The output of that command is a text file including contents of signatures. Each app has a file including information about the developer, and the Linux command we used extracts the contents of that file.

After we extracted signatures, we applied methods from normal Python text information extraction libraries in order to extract necessary information from the signature content and create necessary features.

On top of the package name and developer reputation sub-system introduced in the previous chapter, we have one more reputation sub-system based on elements in AndroidManifest files. In order to parse AndroidManifest files and create reputations, we used APKParser.jar to extract the AndroidManifest.xml files from the apps. We ran the command from a Linux terminal and then used our Python AndroidManifest parser to extract the elements from the XML file. The “xml.etree.ElementTree” library in Python provided an efficient approach for XML information parsing. The reputation calculation was done at the same time as the process for calculating the reputation of the signature and package names; we calculated the benign ratio of each element and stored them into the system.

When we were extracting and processing the features, we also used Python parsers (for timestamp and size features extraction) with “zipfile” and “datetime” libraries. With the help of the Python “zipfile” library, we extracted the time and size information from Android apps.

For tools detection, we applied the Linux zipinfo command and a Python text information parser for analyzing the information from the output of the zipinfo command. Linux zipinfo command outputs structures of the apps and our Python parser is just a text parser used for parsing basic text contents. When we tried to detect patterns for different tools, we found some interesting points. In the apps made by different tools, the structures were different.

If the developer decided to use keytools to sign his APKs, the structure is as shown in Figure 6.1; the red boxes content in the Figure indicates the unique patterns for apps signed using keytool.

```

Zip file size: 7410870 bytes, number of entries: 481
-rw---- 2.0 fat 40583 dU defN 12-Feb-05 16:27 META-INF/MANIFEST.MF
-rw---- 2.0 fat 40704 bL defN 12-Feb-05 16:27 META-INF/BERSERK.SF
-rw---- 2.0 fat 1412 bL defN 12-Feb-05 16:27 META-INF/BERSERK.RSA
-rw---- 2.0 fat 6053 bX defN 12-Feb-05 17:23 assets/app_func_theme.xml
-rw---- 2.0 fat 22024 bL defN 12-Feb-05 17:23 assets/appfilter.xml
-rw---- 2.0 fat 10629 bL defN 12-Feb-05 17:23 assets/desk.xml
-rw---- 2.0 fat 10623 bL defN 12-Feb-05 17:23 assets/drawable.xml
-rw---- 2.0 fat 229 bL defN 12-Feb-05 17:23 assets/menu_item_background.xml
-rw---- 2.0 fat 298 bL defN 12-Feb-05 17:23 assets/preview_del_btn_selector.xml
-rw---- 2.0 fat 301 bL defN 12-Feb-05 17:23 assets/preview_home_btn_selector.xml
-rw---- 2.0 fat 238 bL defN 12-Feb-05 17:23 assets/themecfg.xml
-rw---- 2.0 fat 640 bL defN 12-Feb-05 17:23 res/layout/main.xml
-rw---- 2.0 fat 5224 bL defN 12-Feb-05 17:23 AndroidManifest.xml
-rw---- 1.0 fat 65324 b- stor 12-Feb-05 17:23 resources.arsc
-rw---- 1.0 fat 14500 b- stor 12-Feb-05 17:23 res/drawable-hdpi/advancedtaskkiller.png

Zip file size: 12510915 bytes, number of entries: 109
-rw---- 2.0 fat 9219 bL defN 13-Dec-21 07:01 META-INF/MANIFEST.MF
-rw---- 2.0 fat 9340 bL defN 13-Dec-21 07:01 META-INF/ROHANIEL.SF
-rw---- 2.0 fat 1367 bL defN 13-Dec-21 07:01 META-INF/ROHANIEL.RSA
-rw---- 2.0 fat 1545 bX defN 13-Dec-21 07:01 assets/www/assets/css/style.css
-rw---- 1.0 fat 3525 b- stor 13-Dec-21 07:00 assets/www/assets/img/back-bg.jpg
-rw---- 1.0 fat 79308 b- stor 13-Dec-21 07:00 assets/www/assets/img/bodybg.png
-rw---- 1.0 fat 187434 b- stor 13-Dec-21 07:00 assets/www/assets/img/data-0.png
-rw---- 1.0 fat 336748 b- stor 13-Dec-21 07:00 assets/www/assets/img/data-1.png
-rw---- 1.0 fat 281498 b- stor 13-Dec-21 07:00 assets/www/assets/img/data-10.png
-rw---- 1.0 fat 187117 b- stor 13-Dec-21 07:00 assets/www/assets/img/data-11.png
-rw---- 1.0 fat 348060 b- stor 13-Dec-21 07:00 assets/www/assets/img/data-12.png
-rw---- 1.0 fat 373889 b- stor 13-Dec-21 07:00 assets/www/assets/img/data-13.png
-rw---- 1.0 fat 169068 b- stor 13-Dec-21 07:00 assets/www/assets/img/data-14.png
-rw---- 1.0 fat 200650 b- stor 13-Dec-21 07:00 assets/www/assets/img/data-15.png

Zip file size: 1627209 bytes, number of entries: 15
-rw---- 2.0 fat 998 bL defN 11-May-24 15:16 META-INF/MANIFEST.MF
-rw---- 2.0 fat 1119 bL defN 11-May-24 15:16 META-INF/APPYCITY.SF
-rw---- 2.0 fat 1318 bL defN 11-May-24 15:16 META-INF/APPYCITY.RSA
-rw---- 1.0 fat 39745 b- stor 11-May-24 11:16 assets/20040414170424.jpg
-rw---- 1.0 fat 289437 b- stor 11-May-24 11:16 assets/5DF1rstShoot.png
-rw---- 1.0 fat 19339 b- stor 11-May-24 11:16 assets/94259-004-64570B0A.jpeg
-rw---- 1.0 fat 23803 b- stor 11-May-24 11:16 assets/SantaMonIcaPierSantaMonIcaCalifornia.jpeg
-rw---- 1.0 fat 7270 b- stor 11-May-24 11:16 assets/headerFun1.jpg
-rw---- 1.0 fat 5524 b- stor 11-May-24 11:16 assets/headerLocation1.jpg
-rw---- 1.0 fat 23110 b- stor 11-May-24 11:16 assets/santapier_lcon_114.png
-rw---- 1.0 fat 9445 b- stor 11-May-24 11:16 assets/tiny_light_logo.png
-rw---- 1.0 fat 20467 b- stor 11-May-24 11:16 res/drawable/ya.png
-rw---- 2.0 fat 2552 bL defN 11-May-24 11:16 AndroidManifest.xml
-rw---- 2.0 fat 2688000 bL defN 11-May-24 11:16 classes.dex
-rw---- 2.0 fat 588 bL defN 11-May-24 11:16 resources.arsc

```

Figure 6.1: Structure for apps signed by keytool

If the developer decided to use Jarsigner to sign his APKs, the structure of the apps can be depicted as seen in Figure 6.2. The red boxes include the unique pattern for apps signed with jarsigner.

The structures of apps signed by different tools are included in Appendix B.

In our research, we experimented with J48, Random Forest, SVM and NaiveBayes algorithms. For features selection, we applied the information gain algorithm. All of our features are meta-data and our main goal was to find out if meta-data features are helpful for triaging. In our experiment, we applied the information gain algorithm to help us with feature selection. “Information gain is the expected reduction in entropy caused by partitioning the examples according to a given attribute” [28]. In our work, we use Shannon’s entropy calculation.

```

Zip file size: 4288906 bytes, number of entries: 1029
-rw---- 2.0 fat 100753 [BX] defN 14-Oct-17 17:15 META-INF/MANIFEST.MF
-rw---- 2.0 fat 100806 [BL] defN 14-Oct-17 17:15 META-INF/CERT.SF
-rw---- 2.0 fat 881 [BL] defN 14-Oct-17 17:15 META-INF/CERT.RSA
-rw---- 1.0 fat 5074 b- stor 14-Oct-17 16:56 assets/module/m4533946.png
-rw---- 1.0 fat 1745 b- stor 14-Oct-17 17:15 res/drawable-ndp1/tap_biggrin.png
-rw---- 1.0 fat 331 b- stor 14-Oct-17 17:15 res/drawable-hdpt/holo_dark_navigation_cancel.png
-rw---- 1.0 fat 1729 b- stor 14-Oct-17 17:15 res/drawable-ndp1/tap_blink.png
-rw---- 1.0 fat 1619 b- stor 14-Jul-17 09:50 assets/html/forum/smlays/tap_plnch.png
-rw---- 1.0 fat 440 b- stor 14-Oct-17 17:15 res/drawable-ndp1/abs_ic_commit_search_apt_holo_light.png
-rw---- 1.0 fat 1187 b- stor 14-Oct-17 17:15 res/drawable-ndp1/common_signin_btn_icon_disabled_light.9.png
-rw---- 2.0 fat 508 [BL] defN 14-Oct-17 17:15 res/layout-v14/sherlock_spinner_item.xml
-rw---- 1.0 fat 5738 b- stor 14-Oct-17 17:15 res/drawable-xxhdpt/common_signin_btn_text_disabled_light.9.png
-rw---- 1.0 fat 1208 b- stor 14-Oct-17 17:15 res/drawable-hdpt/lc_bar_feed.png
-rw---- 1.0 fat 154 b- stor 14-Oct-17 17:15 res/drawable-hdpt/actionbar_shadow.9.png

Zip file size: 4696897 bytes, number of entries: 208
-rw---- 2.0 fat 18826 [BX] defN 13-Jun-01 22:45 META-INF/MANIFEST.MF
-rw---- 2.0 fat 18879 [BL] defN 13-Jun-01 22:45 META-INF/CERT.SF
-rw---- 2.0 fat 921 [BL] defN 13-Jun-01 22:45 META-INF/CERT.RSA
-rw---- 1.0 fat 1232 b- stor 13-Jun-01 22:45 res/drawable-hdpt/popupmenu_multiple_choice.png
-rw---- 1.0 fat 133 b- stor 13-Jun-01 22:45 res/drawable-hdpt/translucent_background.9.png
-rw---- 1.0 fat 1164 b- stor 13-Jun-01 22:45 res/drawable-hdpt/menu_rb_bg.9.png
-rw---- 1.0 fat 16813 b- stor 13-Jun-01 22:45 res/drawable-hdpt/dock_sms_part_2.png
-rw---- 1.0 fat 3417 b- stor 13-Jun-01 22:45 res/drawable-hdpt/next_logo.png
-rw---- 1.0 fat 448 b- stor 13-Jun-01 22:45 res/drawable-hdpt/wallpaper_lcon_bg.9.png
-rw---- 1.0 fat 1592 b- stor 13-Jun-01 22:45 res/drawable-hdpt/resize_bg.9.png
-rw---- 1.0 fat 4339 b- stor 13-Jun-01 22:45 res/drawable-hdpt/appdrawer_show_options_light.png
-rw---- 1.0 fat 6701 b- stor 13-Jun-01 22:45 res/drawable-hdpt/common_email.png
-rw---- 1.0 fat 950 b- stor 13-Jun-01 22:45 res/drawable-hdpt/indicator_slide_scroll_end.9.png
-rw---- 1.0 fat 1048 b- stor 13-Jun-01 22:45 res/drawable-hdpt/workspace_trashcan_slide.9.png
-rw---- 1.0 fat 23465 b- stor 13-Jun-01 22:45 res/drawable-hdpt/rotate_icon_chassis.png
-rw---- 2.0 fat 620 [BL] defN 13-Jun-01 22:45 res/drawable/appdrawer_return_workspace_selector.xml

Zip file size: 3372298 bytes, number of entries: 417
-rw---- 2.0 fat 38896 [BX] defN 14-Sep-20 02:20 META-INF/MANIFEST.MF
-rw---- 2.0 fat 38949 [BL] defN 14-Sep-20 02:20 META-INF/CERT.SF
-rw---- 2.0 fat 994 [BL] defN 14-Sep-20 02:20 META-INF/CERT.RSA
-rw---- 1.0 fat 363 b- stor 14-Sep-20 02:20 res/drawable-hdpt/lc_bar_add_on.png
-rw---- 2.0 fat 740 [BL] defN 14-Sep-20 02:20 res/drawable-hdpt/media_player_play_button_selector.xml
-rw---- 1.0 fat 195 b- stor 14-Sep-20 02:20 res/drawable-hdpt/bg_nav_bar.9.png
-rw---- 1.0 fat 1524 b- stor 14-Sep-20 02:20 res/drawable-hdpt/lc_menu_revert.png
-rw---- 2.0 fat 268 [BL] defN 14-Sep-20 02:20 res/raw/gesture_feeditendetail
-rw---- 1.0 fat 771 b- stor 14-Sep-20 02:20 res/drawable/playlist_star_on.png
-rw---- 1.0 fat 350 b- stor 14-Sep-20 02:20 res/drawable/download_on.png
-rw---- 1.0 fat 293 b- stor 14-Sep-20 02:20 res/drawable-hdpt/lc_media_playlist.png
-rw---- 1.0 fat 1187 b- stor 14-Sep-20 02:20 res/drawable-ndp1/common_signin_btn_icon_disabled_light.9.png
-rw---- 1.0 fat 1661 b- stor 14-Sep-20 02:20 res/drawable/btn_check_on.png

```

Figure 6.2: Structure for apps signed by jarsigner

For instance, dataset S includes a number of attributes. When an attribute A splits the set S into subsets S_i , we compute the average entropy and then we compare the sum to the entropy of the original set S . For example we have n different classes, and the entropy is calculated using the following formula. (p_i is the proportion of examples in S belonging to i -th class)

$$-p_1 \log p_1 - p_2 \log p_2 \dots - p_n \log p_n = - \sum_{i=1}^n p_i \log p_i \quad (6.1)$$

In order to compute the average information we need the formula:

$$I(S, A) = \sum_i^n \frac{|S_i|}{|S|} E(S_i) \quad (6.2)$$

So information gain for each attribute A is calculated with this formula:

$$Gain(S, A) = E(S) - I(S, A) = E(S) - \sum_i^n \frac{|S_i|}{|S|} E(S_i) \quad (6.3)$$

With the information gain algorithm, we are able to get a list of attributes with their

entropy identifying which ones are more suitable for classification. (The equations are referenced in [27]) For our work, we wanted to find out the exact number of features that could provide the best accuracy but based on the large number of features included in our framework(380 totally), we decided to try different blocks of the features in our work. For example, based on the information gain algorithm, each attribute was calculated and the algorithm would generate a score and we tried different thresholds to separate the features into feature blocks (each block includes similar a number of features). After that we tried to use blocks of features in our work instead of selecting and removing features one by one. For each different feature group, the number of features is different and the scores of features calculated from information gain are also different, so we included different thresholds for features from each feature group.

We used J48 decision tree for classification. “A decision tree is a decision support system that includes a tree graph helping researchers make decisions. Decision trees are the most powerful approaches in knowledge discovery and data mining” [19]. They are flexible and work with classification problems. In our work, we only needed to provide our data to the algorithm and it would build a classifier based on our data, which was convenient for our work. We also experimented with SVM, Random Forest and NaiveBayes machine learning algorithms.

SVM (Support Vector Machines) is one of the machine learning algorithms we want to include in our work. The algorithm is based on creating hyper planes for classifying data. “A decision plane is one that is separate between a set of objects having different class memberships” [13]. SVM includes binary classifications. SVM aims to find boundaries between outputs and figures out the most optimal way to separate users’ data. The goal of our framework is triaging, so we decided to include this as one of the approaches for our experiment. SVM provides high accuracy and it is also an algorithm that is robust for over-fitting, but the disadvantage of SVM is

also obvious: it is also memory intensive, hard to interpret and expensive in terms of time consumption.

Another algorithm we wanted to include in my experiment is Random Forest. A Random Forest consists of an arbitrary number of simple decision trees; the ensemble of simple trees vote for the most popular class. Random Forests are combinations of tree predictors and a random number of values are included in each random vector. The main idea for this algorithm is to combine those “weak learners” to construct a “strong learner”. Single decision trees often have some disadvantages such as high bias, and random forests are aiming to reduce the effects of those disadvantages. The benefits of random forests are various: they provide high accuracy and run efficiently for large amounts of data. In our framework, we include a large number of apps and more than 300 features, and based on the advantages of Random Forest algorithm we decided to include this algorithm for the classification work.

Another machine learning algorithm we used for classifying my dataset is NaiveBayes. “The Naive Bayes Classifier technique is based on the so-called Bayesian theorem and is particularly suited when the dimensionality of the inputs is high. Despite its simplicity, Naive Bayes can often outperform more sophisticated classification methods” [8]. We chose Naive Bayes because of its excellent performance. It is extremely fast compared to other algorithms; based on the performance aspect it is an outstanding option. In Naive bayes, the algorithm assumes each feature is independent from each other but its main disadvantage is that it cannot figure out the interactions between features. This property of the algorithm seems not suitable for our work because some of the features used in our framework were calculated based on other features so there are many connections between different features but still, based on its lower time and resource consumption, we decided to include this. In our experiment, our main focus was to extract and test our meta-data features to determine whether they can be used for malware triaging; meanwhile we used some

already existing machine learning algorithms and tools to run the experiment; our focus was not on the design of new classification algorithms.

6.2 Experiment setup

The experiments to evaluate our system are carried out on an Intel Core i7-4790 CPU @ 3.60GHz machine with 15.6 GB RAM. The classifiers are evaluated with a 10-fold cross-validation. In a 10-fold cross-validation, the dataset is split into 10 subsets and the work iterates 10 times. For each iteration, one subset is a test set and all the rest are used for training. For this approach each instance is included at least once in the test set and nine times in the training sets.

For some feature processing, we applied statistical analysis. Among all features, we included intervals as parameters used for timestamp features and reputation feature processing. In this section, we give our explanation of the parameters design and value selection.

During processing the raw features, we applied statistical analysis. We analyzed the features based on different time intervals (see Table A.2). The settings of our intervals were based on our previous estimates; at the beginning, we had a list of time intervals: **1day** indicates the number of files created just in 1 day before the app uploaded to market, **7days** indicates the number of files created in a week before the app upload and similarly **1month**, **3month**. We tried different time intervals in preliminary experiments and we noticed these intervals gave us the best accuracy. From some apps we had before, we determined the timestamp of the files in each app have various timestamps; some of them may have been created a long time ago compared to the app creation time and some files have the same timestamp as the app creation time. We simply assumed those time intervals as thresholds. After that, we did some related work and we found out our time intervals are not suitable

for detecting the apps because they are not typical. For different app tools, there are slightly different time gaps between each files, for example, some can be 2 minutes or longer but still less than 1 hour. So we thought our beginning assumption was infeasible. We modified our time intervals from the original ones to the new ones, including **2mins** indicating the number of files created just 2 minutes before the app was made, **2hours** standing for the number of files created in 2 hours before the app was made; similarly we have other time intervals including **1 day**, **1 month** and **longer**.

When we tried to extract features related with the reputations, we applied the same work as we did for the timestamp raw features: we set thresholds and calculated the contents located at each reputation interval. We set 5 intervals; we call them reputation intervals, they are here to represent the reputations. At the beginning, we decided to use 0, 0.2, 0.4, 0.6, 0.8 and 1; these numbers stand for the reputations of each element in the reputation system. We did not split them into more intervals because we needed to consider the time consuming aspect of our experiment. And after that, we tried to use different reputation intervals and we figured out a better interval(1, 0.1, 0.4, 0.6, 0.8, 1) which provided higher accuracy. These intervals provided the best accuracy in our experiment. As for the features extracted from AndroidManifest file in Table A.6, we mainly considered the accuracy of the results before we set the values for our statistical work.

6.3 Experiment results

To evaluate the performance of our approach, we conducted several experiments focusing on the following objectives:

- meta-data features assessment. In this experiment, we aimed to assess individual groups of features and select the most suitable one for accurately classifying

apps.

- Analysis of combination of features. With the combination of meta-data features, we aim to select the best combinations based on all groups and assess the result.
- Performance analysis. When implementing the framework, we tried different approaches and we aim to analyze the performance between the approaches and select the most efficient one.

6.3.1 Assessment of feature group individually

There are some meta-data features who can be efficiently used for malware triage because they provide inspiring accuracy; meanwhile some features showed their potential for other Android analysis but not malware triage. The classification accuracy in experiments was evaluated using traditional evaluation metrics: TPR (true positive rate) and FPR (false positive rate).

Size based feature group

The tool we included in our experiment for fulfilling these works was Weka 3.6.10. Table 6.1 gives classification results for size-based features using different machine learning algorithms. We noticed SVM and Naive Bayes are not suitable for our classification, not only because they provide low accuracy but also the time overhead of SVM is unavoidable, it took 8 hours 16 minutes and 55.2 seconds to finish the classification compared to 2min 15 seconds for Decision Tree, 2min 3 seconds for RandomForest and 47.6 seconds for Naive Bayes. With the purpose of trying to triage more malicious apps from the dataset and with less overhead, we chose J48 (Decision Tree) for the rest of our experiment.

There are in total 45 size features and after we applied the information gain attribute

Algorithm	Class	TP rate	FP rate
J48	benign	0.947	0.512
	malicious	0.488	0.053
Random Forest	benign	0.98	0.567
	malicious	0.433	0.02
SVM	benign	0.996	0.975
	malicious	0.025	0.004
NaiveBayes	benign	0.926	0.906
	malicious	0.094	0.074

Table 6.1: Size based features classification based on different machine learning algorithms

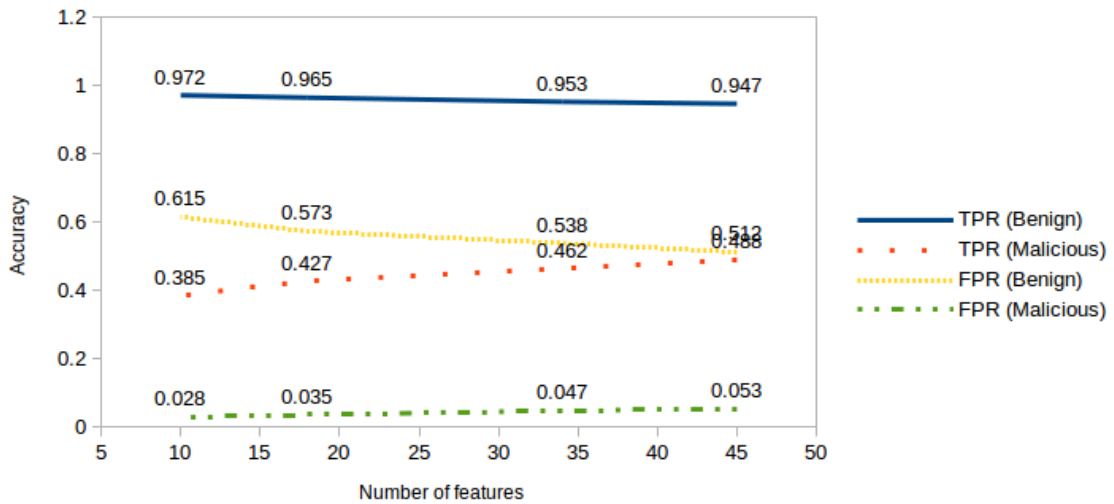


Figure 6.3: Classification based on selected size features

selection algorithm and we selected different numbers of the top features and checked their classification performance. From Figure 6.3, we got the highest malicious apps TP rate with all 45 features included for machine learning algorithm and 48.8% of the malicious apps were correctly classified. With the decreasing number of features selected, fewer and fewer malicious apps were correctly classified but more and more benign apps were classified correctly. At this point, we focused on filtering out as many malicious apps as possible so from the detailed table, at most 48.8% of malicious apps can be correctly filtered out.

As a meta-data feature, size provides promising results. With only size features, at

most 48.8% of the malicious APKs are correctly classified. Even though the accuracy is low, it provides the potential of the features; size can be used as a feature, and based on size features only, we were able to classify almost half of the malicious ones which is encouraging.

Timestamp based feature group

In total we did an extraction of 148 timestamp based features. From Figure 6.4 we concluded that the classification result does not change much based on the number of features. Classification based on 81 timestamp features provided slightly better results compared to the others. With timestamp features only, we were able to correctly label 30% of malicious apps. Timestamp-based features alone are not efficient for malware triage; however due to low FPR for malware apps they can be used for assistance in other analysis of malicious apps.

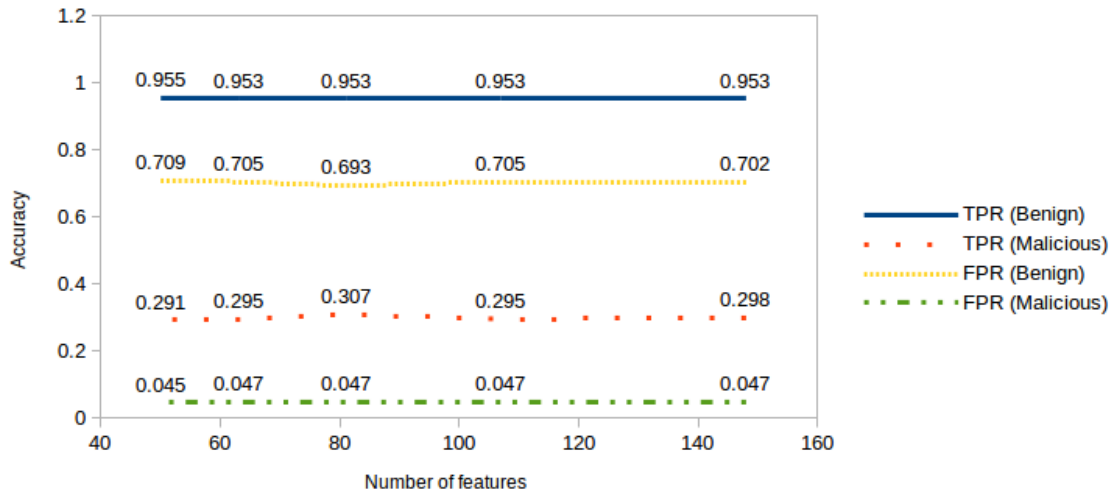


Figure 6.4: Classification based on selected timestamp features

Third party library & AndroidManifest file analysis

We included features extracted from AndroidManifest files, including the analysis for reputation of the element.

In this feature group, we collected 199 features based on the elements from Android-

Manifest files combined with their reputation value. The classification results did not vary much with the number of features (see Figure 6.5); however the highest classification (93.63%) is based on 174 features.

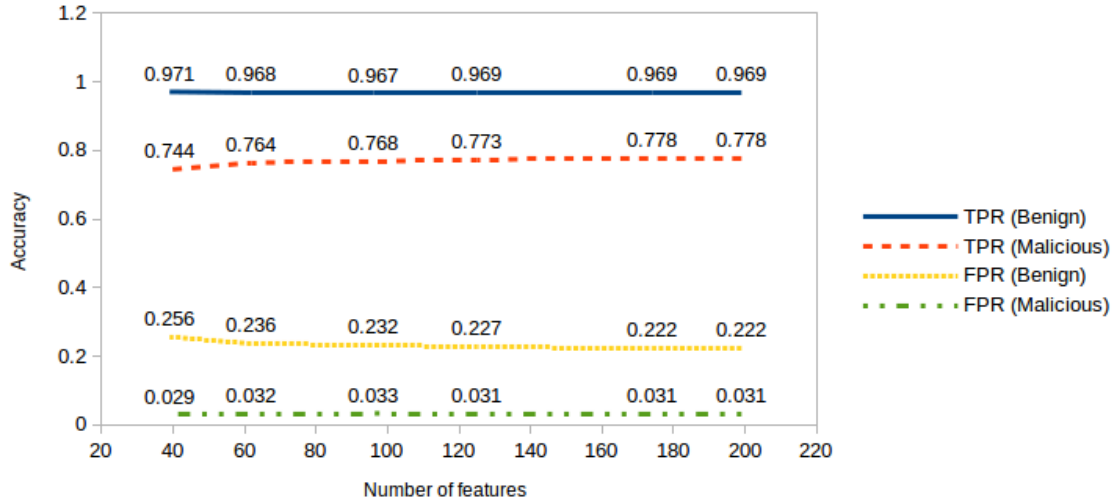


Figure 6.5: Classification based on selected features from AndroidManifest file

From the result, we noticed that features extracted from AndroidManifest file combined with reputation provides a positive result. The reputation system played an important role in our work. This is an encouraging finding, reputation represents a lot of information and assistance in malware triaging. Meanwhile, features extracted from AndroidManifest file always have relations to Android app content so this feature includes more information compared to other features such as size or timestamp. The design of the reputation system is the most important framework in our design. As it was proved, meta-data features can be used efficiently for malware triaging, 77.8% of malware were correctly classified; this is a great progress in malware triaging using meta-data features.

There are multiple analysis' using permissions. We use our reputation system for permission features and according to permission-based features extracted in our work, we have triaging results in Table 6.2. We noticed using only permission-based features It is not able to classify malware efficiently so we added other materials to our

framework.

Class label	TPR	FPR
Benign	95%	41.5%
Malicious	58.5%	5%
Overall	88.73%	11.27%

Table 6.2: Triaging with only permission-based features

Developer & package name reputations-based features classification

In our work, we not only included reputation for elements from AndroidManifest files but also for authors and package names. In these features, we only have 9 features so we did not apply any feature selection techniques. The result of classification based on authors and package names are in Table 6.3.

All of the features in this feature group are reputation based features but for authors and package names. As a meta-data feature group, they are encouraging for classification although far from adequate. Almost half of the malicious apps can be classified correctly and these features should be combined with others for future analysis.

Tool-based features

The last feature group is related with the tool used for apps combined with signature information analysis. There are only 4 features in this feature group (see Figure 6.4) so we did not apply any feature selection works.

The results from this are disappointing but expected. Table 6.5 shows a detailed result of classification for different types of tools.

Our analysis based on tools detection is not complete because of the limitation of the number of apps from different tools. Currently, we can only detect the apps developed by current editions of the following tools: Adobe Air, Keytool, JarSigner, Dot42, Eclipse and Android Studio. There are still some other tools such as AppInventor, etc. We found out the structure (we use of zipinfo command in Linux to extract the structure of an app) for some apps have patterns, such as the certification files

Number of features	Class	TP rate	FP rate
9	benign	0.974	0.51
9	malicious	0.49	0.026

Table 6.3: Detailed classification based on reputation features

Number of features	Class	TP rate	FP rate
4	benign	0.997	0.977
4	malicious	0.023	0.003

Table 6.4: Detailed classification based on tools features

located at the beginning and at the bottom in the output files from the zipinfo command, so we assume they are from other tools however we are yet to detect them. As a result in Table 6.5, it shows the apps signed by Keytool or Jarsigner are much more likely to be benign than those signed with other tools. We assume this is because of the apps repackaging. Usually, when attackers decompile apps and sign their author information; they choose these two ways. However, further analysis is required here. At the same time, we provide some assistance for future work based on Android tools detection rather than triage.

6.3.2 Classification based on combination of features

Finally, we combined all 380 features from all different feature groups. Our classification results are based on the features selected by information gain. The results are shown in Figure 6.6.

With the combination of all features we achieved optimistic classification results: 80% malicious apps and 97% benign apps were correctly classified. The best classification result is based on 300 out of the total 380 features. However, with 122 features, we use much less time overhead compared to using more features. As Figure 6.6 shows the decrement of feature numbers does not have a huge effect on classification accuracy. As a conclusion, we can claim meta-data features are able to assist with malware

Tool name	Number of benign apps	Number of malicious apps	Number of total apps	Benign ratio
Adobe air	1188	65	1253	94.89%
Keytool	5314	1879	7193	73.88%
BottomCerts	9	0	9	100%
Dot42	79	7	86	91.86%
Eclipse/Android Studio	6509	747	7256	89.71%
JarSigner	1459	688	2147	67.96%
Someware	17869	3244	21113	84.64%
TopCerts	413	206	619	66.72%
Total	32840	6836	39676	82.78%

Table 6.5: Benign ratio of apps made by different tools

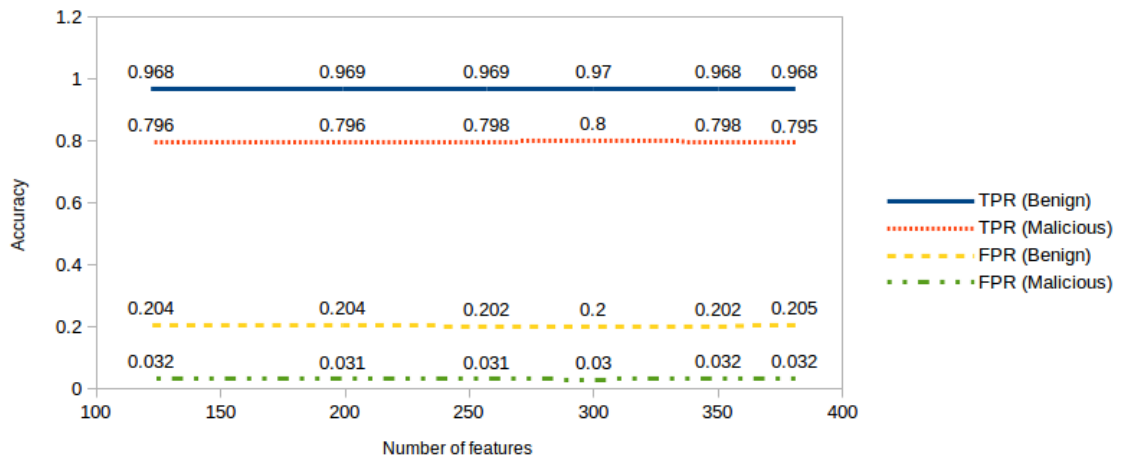


Figure 6.6: Classification based on selected combination features

triage. Simply based on meta-data features without any information extraction from byte/source code, without any other meta-data from Android markets, we have the ability to get an accuracy of 80% TP rate of malware and 97% TP rate of benign apps which is encouraging. From our work, we provided a reputation system containing reputation values for different elements from files and this reputation provided lightweight features but impressive accuracy. Our experiment includes apps from different markets from across the entire globe with apps collected from different years so our work can represent today's Android app environment. Again, all of the

features in our work have no relation to byte/source code, so the whole system is obfuscation resistant. For the tool detection section, we did not get prospective accuracy but based on our work, we provided helpful meta-data features for tools detection and we also provided a likely distribution of malware based on different tools from our work which is helpful for future research.

6.3.3 Chain classification

Besides all the classification work we did above, we also used chain classification in order to triage malware step by step using different meta-data features. In each step, we applied different meta-data features to triage malware by labelling them benign or malicious; we filtered out the ones labelled as malicious and passed the rest to the next step. Based on our results, we got the highest classification accuracy using the features extracted from AndroidManifest files so we used it for the first step. After the first step we used size features because it is the second most efficient feature group, but based on our preliminary work we did not triage many apps using size features after using AndroidManifest features in the first step. Therefore we tried two different models: keeping chain classification using different feature groups (see Figure 6.7) and combining all the rest of the features as one for the second step (see Figure 6.8).

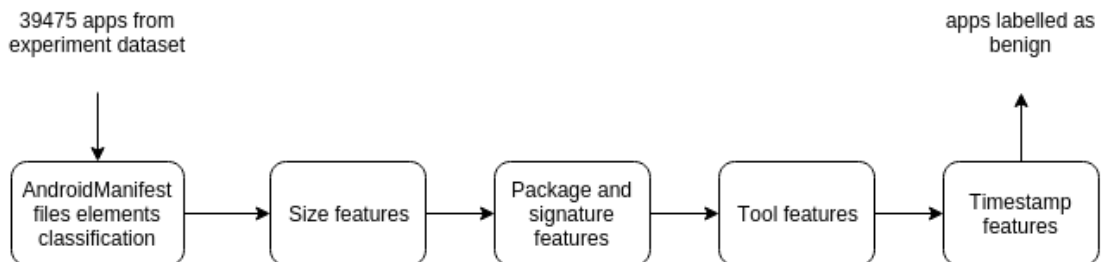


Figure 6.7: Chain classification model 1

Based on the concern of time/resource overhead, we used 39475 apps from our ex-

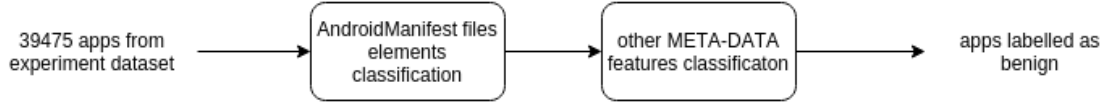


Figure 6.8: Chain classification model 2

periment dataset. For the first model, the number of triaging apps from each step are shown in Table 6.6.

Step number	Features included	Malicious apps triaged at each step (correctly classified)	Benign apps triaged (misclassification)	Number of apps triaged at each step
1	Android Manifest files elements	5932	282	6214
2	Size features	124	9	133
3	Package & signature features	42	18	60
4	Tool features	5	0	5
5	Timestamp features	0	0	0
Sum	All	6103	309	6412

Table 6.6: Triaging details

After our chain classification, we finally triaged 6412 apps that contained 309 benign apps (misclassified) and 6412 malicious apps.

For the second model, details of triaging using chain classification step by step is shown in the following list:

- All 39475 apps are inserted into the first step which used AndroidManifest meta-data features for triaging. In the 39475 apps, 32674 are benign and 6801 are malicious.
- In step 1, 6224 apps were labelled as malicious. Among them, 282 benign apps were misclassified and 5942 malicious apps were correctly labelled. We labelled

6224 apps as malicious; after filtering them out we then passed the rest on to the next classifier.

- In the second step we combined all of the remaining meta-data features. Because each separate feature is too weak to classify malware, we combined them together as a single strong one. After doing this, 212 apps were labelled as malicious. Among them 14 benign apps were misclassified and 198 malicious apps were correctly classified.
- The chain classification contains only two steps; after that, 6436 apps were triaged and 33039 apps were left and labelled as benign. According to the final classification, 32378 are benign and 661 are malicious.

6.3.4 Experiments with Drebin dataset

In order to test the robustness of our framework, we tried to triage malware from Drebin dataset using our approaches. Drebin dataset is a benchmark dataset which includes malware samples from different malicious families. The dataset is used by a large number of studies [6]. Until recently it was seen as a benchmark dataset in the security community. Based on our analysis of the samples included in the dataset, there are also some drawbacks for this dataset:

- The dataset includes outdated malware: This dataset is a widely used dataset because it includes malware for various malware families but it is not suitable for current malware classification. The dataset was created several years ago with some formerly malware samples that no longer exist on today's markets. Based on our research, a large number of apps from markets now are adware but many samples from Drebin dataset belong to adware families.
- The samples in Drebin dataset are manually selected: In order to create a dataset that includes apps from malware families for analysis, it is necessary to

include a proper number of apps from each family but the number of malicious apps from families in online app stores are different. If a dataset is created with manually selected apps then this dataset is not the best option for representing real life app stores.

- All apps in Drebin dataset are malware: It is a dataset including only malicious apps so there is no comparison between the malicious apps from each different family and benign apps. Thus this dataset is not suitable for binary classification because there is no information that can be extracted for benign apps.

Using our framework, we finally triaged 77% of the malware from Drebin dataset using the combination of our meta-data features. The result is close to the experiment based on our own dataset; we got 80% malware triaged from our dataset so the framework we established is robust for over-fitting.

6.3.5 Comparison with another static analysis tool

Most of the studies for Android malware classification use static analysis; in order to compare the time/resource overhead and accuracy of our framework with those of some static tools, we use AndroWarn to scan the apps in our dataset.

Androwarn [4] is a static tool for Android application code analysis. The detection is based on the Dalvik bytecode of Android applications and after scanning there will be a report sent back to the users. We compared the performance of our framework and Androwarn along the following points:

- Robustness: More than 20% of the apps in our dataset have errors when running Androwarn tool and there are no proper approaches for exceptions catches in the tool. For static analysis, there is always a high risk of failure for using reverse engineering techniques to extract the source code. So there must

be some measures to deal with the errors when using static analysis tools for Android applications.

- Time overhead: It took more than 5 days to finish scanning all apps (39475 apps) in our experiment using Androwarn. But Androwarn only provides scanning reports instead of binary classification results, so we created some features (40 features totally) extracted from the report to help us classifying the dataset. With our features parser, it takes 37.7 seconds to parse 6417 apps. We did not parse the result from all reports because of the time usage.
- Results based on the features: With the features extracted from 6417 apps (399 malware, 6018 benign apps), we were able to triage 53% of the malware.

From the results, we noticed the time overhead with Androwarn is much higher than using our framework and there is a high risk of failure using this tool. Using this tool, only scanning reports are sent back to users instead of binary classification results; then there would be more work for the users to extract useful information from their reports and this is also difficult. We did not get promising accuracy with the features we manually extracted.

There are many analysis tools [3]; we tried to work with some other approaches. For most of the tools, users need to spend plenty of time on the configuration work and use proper settings in order to run tools without issues. For most of the static analysis tools, a high risk of failure always exists and they are expensive in terms of time and resource overhead. Complex settings should be set before running the static or dynamic tools, errors and exceptions should be dealt with properly and another disadvantage for binary classification is that they only provide scanning reports and leave the judgement to the researchers.

6.3.6 Performance analysis

In our experiment, we created a framework with manually created tools and scripts to extract features. Based on the materials we included in our framework the performance can be judged in two ways: “robustness” that measures whether our framework can deal with all exceptions a user may confront and “time/resource consuming” that represents the overhead.

Robustness

All of our feature parsers are able to deal with exceptions during processing time. There are exceptions from our parsers when we tried to extract features, mainly because of the structures of the apps. Using special tools to package or repackage apps, applied with some obfuscation or protection techniques may cause app structure that is complex and impossible for extracting data. Our measure is to add default values to those features. Essentially, our framework will not crash once you start to process.

Time/resource overhead

Our framework was built using experimental setup given in Section 6.2. For an APK with a size of 101.0 MB, processing time of different feature parsers in our framework are shown in the following list:

- Size parser: 0.069 seconds
- Timestamp parser: 0.067 seconds
- AndroidManifest file data parsing: 0.42 seconds
- Package name reputation parsing: 1.53 seconds
- Developer reputation parsing: 0.033 seconds
- Tool features parser: 0.043 seconds

The overhead time listed above are based on each different feature-based parser. Currently we are trying to apply multi-threading techniques to our parsers in order

to reduce the processing time. However, some parsers for extracting different features can possibly be merged together such as a timestamp and size features parser.

In order to test the performance, we include 39710 instances with all 381 meta-data combined features for a J48 machine learning algorithm. 16 minutes and 50 seconds are used for classification, if we applied feature selection algorithms to our work, consuming time will be decreased because of the decrement of feature number (see Table 6.7).

Number of features	Time overhead
380	1010 seconds
350	875.5 seconds
300	841.3 seconds
257	547.6 seconds
199	521 seconds
122	234.1 seconds

Table 6.7: Time overhead with number of features included

Chapter 7

Discussion

In our work, we use pure meta-data feature groups for Android malware triaging. Each feature group contains a list of meta-data features extracted directly from the apps in our manually built dataset. With all meta-data features included for triaging, our approach was able to achieve 80% detection rate with only a 3% false classification rate. We finished lightweight classifying large amounts of unknown apps and used less time.

We got our best classification result with 80% TPR on malicious apps based on 300 meta-data features out of 381 in total. However, from those selected 122 features, the distribution of features from a different feature group is shown in Table 7.1.

Name of feature group	Number of features selected	total number of features
Size	18	45
Timestamp	0	148
AndroidManifest file elements	102	174
Package name & certificates reputation	1	9
Tool	1	4

Table 7.1: Detailed classification based on tools features

From Table 7.1 we noticed that features extracted from AndroidManifest files have

reputations with the biggest portion. In that feature group, elements from Android-Manifest files are used to create features. The reputation system based on elements from AndroidManifest files provides reliable support on malware triaging compared to other types of features. In the 122 features, there is no timestamp-based features involved, they are not practical for malware triaging. 18 out of 45 size-based features are selected in the 122 combinations group compared with other features, they are able to provide assistance on malware triaging. All 122 features are processed with count or statistical methods and they are able to have a 80% TPR and only 3% FPR for malicious apps classification. The selected 122 features are marked in the tables in Appendix based on their different feature group.

In related work, we introduced some researches using only permission-based features for malware analysis. Even though the majority of studies in this field claim the high accuracy of analysis is based on permission analysis only, we were only able to achieve 58.5% malware correctly classified in our experiment, using permissions-based features only. It was not able to get excellent accuracy in our framework.

We provide explainable detection, package names & certification-based features, size-based features and features based on AndroidManifest file elements these are practical for triaging malicious apps. Some other features such as timestamp-based features could be used for triaging malicious apps as well but the performance is not as good as others. However, none of the features are efficient for benign apps triaging. Generally, there is only a small portion of the apps in one market that are malicious and based on this situation, apps are too easily labelled as benign and most of the malicious apps are misclassified. There is also the limitation of using solely meta-data features, employing meta-data features is only practical to triage a portion of malicious apps but is not practical for triaging benign apps.

Complementing our work with static and dynamic techniques would provide better result. Our work can be used as a pre-filter before researchers apply any static and

dynamic approaches. There will be less time overhead in triaging combined with using a meta-data approach.

From the two different chain classification models we can also triage a large number of malicious apps with low benign apps misclassified. If we use the first model, we can triage 6103 out of 6801 malicious apps (89.7%) and 309 out of 32674 benign apps (1%) are misclassified. If we use the second model, we are able to classify 6140 out of 6801 malicious apps (90.3%) and 296 out of 32674 benign apps are misclassified (1%). Using chain classification, we can also triage malware efficiently.

Several other points we need to discuss:

- For our features, we used different techniques to process the raw features, but mainly we split them into different intervals. For timestamp features, we processed them based on time intervals and for size features, we processed them based on their types, there could have been more approaches to feature processing which may create better classification results.
- Tools detection features are included in our work. We built a lightweight approach to analyze the tool in order to detect the usage of several development tools. Although the pattern is not robust yet, the pattern researchers can do much more/deeper research in the future.
- Our result however is more realistic as it is based on a diverse and recent dataset collected from 8 markets across the globe. It also has the largest set used for this type of analysis so far.
- The reputation framework provides the best result; as meta-data features, they showed a good potential for malware triaging. We found that the larger dataset we have and the more reputation information we have, the better classification result we will get. So if this framework is applied to the Android app stores,

people should have a large app dataset for creating the reputations and this provides great help.

- With the help of Linux command “zipinfo” we extracted the structures of apps and we temporarily used them for tools detection. We also noticed that timestamps of the files in the structures could help tools detection which we did not pursue, but it could be used for a potential investigation.
- We crawled and built our dataset in August 2015 from 8 different app stores around the world. Our dataset is the most representative one, it includes apps from developers worldwide. We offer this data to a research community in hopes of improving analysis in this field.
- The framework we built is based on meta-data with the purpose of triaging. This analysis can be further complemented with static and dynamic techniques.

Chapter 8

Conclusion and Future work

8.1 Conclusion

In this work, we presented an approach showing that meta-data features extracted from apps can be used to assist with malware triaging. The framework of meta-data features extraction and the approaches to processing the features are the most important contributions.

The reputation mechanism in our work provides a significant help in malware triaging. This meta-data system provides great opportunities for researchers to have a much better straightforward approach to detecting apps without analyzing the contents.

Besides our main contribution to developing the reputation framework we also have some other important findings. We have demonstrated that size features provide inspiring results; less than close to 50% of malicious apps can be detected, which proved our hypothesis. At the beginning, we were motivated to use size features based on the assumption that “size similar apps would have more content shares”.

In our work, during meta-data features collection we also made some other findings. We found an imperfect approach for detecting the tools used for different apps.

Besides, we also found that in using meta-data features we may have the ability to detect some specific malware; for example, some adwares exhibit some consistent patterns.

As a conclusion, meta-data features can be used to assist malware triaging, although the output is not perfect compared to static or dynamic analysis, the parser for meta-data features extractions is much easier to implement and there is much less installation on third party tools. Also, meta-data features have great potential on different Android analysis aspects.

8.2 Future work

Although we have proved our hypothesis we have found many areas in which our research can be extended as summarized below:

- Researchers can also include meta-data features derived from external sources for example, features from app stores such as download numbers, categories, comments and review numbers for malware triaging. These features are precious resources and can also be used in analysis.
- One of the most important materials in our work is the reputation system establishment. Reputation is calculated based on a large number of apps. Our dataset is the largest that currently exists in research. Further analysis is needed for deciding if the dataset is large enough for building a reputation repository.

References

- [1] *android receivers*, http://www.tutorialspoint.com/android/android_broadcast_receivers.htm.
- [2] *android services*, http://www.tutorialspoint.com/android/android_services.htm.
- [3] *Androtools*, <https://github.com/ashishb/android-security-awesome>.
- [4] *Androwarn*, <https://github.com/maaaaz/androwarn>.
- [5] *Dangerous permission list*, <http://developer.android.com/guide/topics/security/permissions.html>.
- [6] *Drebin dataset*, <https://www.sec.cs.tu-bs.de/danarp/drebin/>.
- [7] *Hardware features*, <http://developer.android.com/guide/topics/manifest/uses-feature-element.html#hw-features>.
- [8] *Naive bayes*, <http://www.statsoft.com/textbook/naive-bayes-classifier>.
- [9] *Permission list*, <http://developer.android.com/reference/android/Manifest.permission.html>.
- [10] *priorities*, <http://developer.android.com/guide/topics/manifest/intent-filter-element.html>.
- [11] *Random forests*, <http://www.statsoft.com/Textbook/Random-Forest>.

- [12] *Software features*, <http://developer.android.com/guide/topics/manifest/uses-feature-element.html#sw-features>.
- [13] *Support vector machines (svm) introductory overview*, <http://www.statsoft.com/Textbook/Support-Vector-Machines>.
- [14] Emanuel Achimescu, Radu Ovidiu Preda, Nicolae Vizireanu, and Ionut Pirnog, *Multiple point android digital image authentication*, Signals, Circuits and Systems (ISSCS), 2015 International Symposium on, IEEE, 2015, pp. 1–4.
- [15] Alessandro Armando, Roberto Carbone, Gabriele Costa, and Alessio Merlo, *Android permissions unleashed*, Computer Security Foundations Symposium (CSF), 2015 IEEE 28th, IEEE, 2015, pp. 320–333.
- [16] Daniel Arp, Michael Spreitzenbarth, Hugo Gascon, and Konrad Rieck, *Drebin: Effective and explainable detection of android malware in your pocket.*, 2014.
- [17] David Barrera, H. Güneş Kayacik, Paul C. van Oorschot, and Anil Somayaji, *A methodology for empirical analysis of permission-based security models and its application to android*, Proceedings of the 17th ACM Conference on Computer and Communications Security (New York, NY, USA), CCS '10, ACM, 2010, pp. 73–84.
- [18] David Barrera, H Güneş Kayacık, PC van Oorschot, and Anil Somayaji, *A methodology for empirical analysis of permission-based security models and its application to android*, (2010).
- [19] Neeraj Bhargava, Girja Sharma, Ritu Bhargava, and Manish Mathuria, *Decision tree analysis on j48 algorithm for data mining*, Proceedings of International Journal of Advanced Research in Computer Science and Software Engineering **3** (2013), no. 6.

- [20] Michael Bierma, Eric Gustafson, Jeremy Erickson, David Fritz, and Yung Ryn Choe, *Andlantis: large-scale android dynamic analysis*, arXiv preprint arXiv:1410.7751 (2014).
- [21] Abhijit Bose, Xin Hu, Kang G Shin, and Taejoon Park, *Behavioral detection of malware on mobile handsets*, Proceedings of the 6th international conference on Mobile systems, applications, and services, ACM, 2008, pp. 225–238.
- [22] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani, *Crowdroid: behavior-based malware detection system for android*, Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, ACM, 2011, pp. 15–26.
- [23] Jonathan Crussell, Clint Gibler, and Hao Chen, *Andarwin: Scalable detection of semantically similar android applications*, Computer Security–ESORICS 2013, Springer, 2013, pp. 182–199.
- [24] dflower, *Google play crawler*, <https://github.com/dflower/google-play-crawler>.
- [25] Ken Dunham, Shane Hartman, Manu Quintans, Jose Andre Morales, and Tim Strazzere, *Android malware and analysis*, 1st ed., Auerbach Publications, Boston, MA, USA, 2014.
- [26] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth, *Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones*, ACM Transactions on Computer Systems (TOCS) **32** (2014), no. 2, 5.
- [27] F.Aiulli, *Entropy and information gain*.
- [28] F.Alolli, *Entropy and information gain*, <http://www.math.unipd.it/aiolli/corsi/0708/IR/Lez12.pdf>.

- [29] Parvez Faruki, Vijay Ganmoor, Vijay Laxmi, M. S. Gaur, and Ammar Bharmal, *Androsimilar: Robust statistical feature signature for android malware detection*, Proceedings of the 6th International Conference on Security of Information and Networks (New York, NY, USA), SIN '13, ACM, 2013, pp. 152–159.
- [30] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner, *Android permissions demystified*, Proceedings of the 18th ACM Conference on Computer and Communications Security (New York, NY, USA), CCS '11, ACM, 2011, pp. 627–638.
- [31] Adrienne Porter Felt, Kate Greenwood, and David Wagner, *The effectiveness of install-time permission systems for third-party applications*, University of California at Berkely, Electrical Engineering and Computer Sciences, Technical report No. UCB/EECS-2010-143 (2010).
- [32] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner, *Android permissions: User attention, comprehension, and behavior*, Proceedings of the Eighth Symposium on Usable Privacy and Security, ACM, 2012, p. 3.
- [33] Yu Feng, Saswat Anand, Isil Dillig, and Alex Aiken, *Apposcopy: Semantics-based detection of android malware through static analysis*, Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, 2014, pp. 576–587.
- [34] Yanick Fratantonio, Aravind Machiry, Antonio Bianchi, Christopher Kruegel, and Giovanni Vigna, *Clapp: characterizing loops in android applications (invited talk)*, Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile, ACM, 2015, pp. 33–34.

- [35] Michael I Gordon, Deokhwan Kim, Jeff H Perkins, Limei Gilham, Nguyen Nguyen, and Martin C Rinard, *Information flow analysis of android applications in droidsafe.*, NDSS, 2015.
- [36] Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang, *Riskranker: scalable and accurate zero-day android malware detection*, Proceedings of the 10th international conference on Mobile systems, applications, and services, ACM, 2012, pp. 281–294.
- [37] IDC, *Smartphone os market share*, <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>.
- [38] Jinseong Jeon, Kristopher K. Micinski, Jeffrey A. Vaughan, Ari Fogel, Nikhilesh Reddy, Jeffrey S. Foster, and Todd Millstein, *Dr. android and mr. hide: Fine-grained permissions in android applications*, Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (New York, NY, USA), SPSM '12, ACM, 2012, pp. 3–14.
- [39] Patrick Gage Kelley, Sunny Consolvo, Lorrie Faith Cranor, Jaeyeon Jung, Norman Sadeh, and David Wetherall, *A conundrum of permissions: installing applications on an android smartphone*, Financial Cryptography and Data Security, Springer, 2012, pp. 68–79.
- [40] Katharyn Kennedy, Richard V Aghababian, Lucille Gans, and C Phuli Lewis, *Triage: techniques and applications in decisionmaking*, Annals of Emergency Medicine **28** (1996), no. 2, 136–144.
- [41] Martina Lindorfer, Matthias Neugschwandtner, and Christian Platzer, *Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis*, Computer Software and Applications Conference (COMP-SAC), 2015 IEEE 39th Annual, vol. 2, IEEE, 2015, pp. 422–433.

- [42] Microsoft, *malware*, <https://technet.microsoft.com/en-us/library/dd632948.aspx>.
- [43] Rene Millman, *Updated: 97Android*, <http://www.scmagazineuk.com/updated-97-of-malicious-mobile-malware-targets-android/article/422783/>.
- [44] Pulse Secure Mobile Threat Center (MTC), *2015 mobile threat report*.
- [45] Alfonso Munoz, Ignacio Martin, Antonio Guzman, and Jose Alberto Hernandez, *Android malware detection from google play meta-data: Selection of important features*, Communications and Network Security (CNS), 2015 IEEE Conference on, IEEE, 2015, pp. 701–702.
- [46] newzoo, *Top 10 android app stores — china*, <https://newzoo.com/insights/rankings/top-10-android-app-stores-china/>.
- [47] NOD32, *about nod32*, <http://www.eset.com/us/?ref=AFC-CJ&attr=7209273&pub=12231003&shop=sv153280>.
- [48] Ramjee Prasad, Károly Farkas, Andreas U Schmidt, Antonio Lioy, Giovanni Russello, and Flaminia Luccio, *Security and privacy in mobile information and communication systems: Third international icst conference, mobisec 2011, aalborg, denmark, may 17-19, 2011, revised selected papers*, vol. 94, Springer, 2012.
- [49] Margaret Rouse, *metadata*, <http://whatis.techtarget.com/definition/metadata>.
- [50] Bhaskar Pratim Sarma, Ninghui Li, Chris Gates, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy, *Android permissions: A perspective combining risks and benefits*, Proceedings of the 17th ACM Symposium on Access Control Models and Technologies (New York, NY, USA), SACMAT '12, ACM, 2012, pp. 13–22.

- [51] Ryo Sato, Daiki Chiba, and Shigeki Goto, *Detecting android malware by analyzing manifest files*, Proceedings of the Asia-Pacific Advanced Network **36** (2013), 23–31.
- [52] Aubrey-Derrick Schmidt, Rainer Bye, Hans-Gunther Schmidt, Jan Clausen, Osman Kiraz, Kamer A Yüksel, Seyit A Camtepe, and Sahin Albayrak, *Static analysis of executables for collaborative malware detection on android*, Communications, 2009. ICC'09. IEEE International Conference on, IEEE, 2009, pp. 1–5.
- [53] Aubrey-Derrick Schmidt, Frank Peters, Florian Lamour, Christian Scheel, Seyit Ahmet Çamtepe, and Şahin Albayrak, *Monitoring smartphones for anomaly detection*, Mobile Networks and Applications **14** (2009), no. 1, 92–106.
- [54] Alisa Shevchenko, *Malicious code detection technologies*, Kaspersky Lab (2008).
- [55] Michael Spreitzenbarth, Felix Freiling, Florian Echtler, Thomas Schreck, and Johannes Hoffmann, *Mobile-sandbox: having a deeper look into android applications*, Proceedings of the 28th Annual ACM Symposium on Applied Computing, ACM, 2013, pp. 1808–1815.
- [56] statista, *Number of available applications in the Google Play Store from December 2009 to February 2016*, <http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>.
- [57] Eric Struse, Julian Seifert, Sebastian Üllenbeck, Enrico Rukzio, and Christopher Wolf, *Permissionwatcher: Creating user awareness of application permissions in mobile systems*, Ambient Intelligence, Springer, 2012, pp. 65–80.
- [58] Peter Teufl, Michaela Ferk, Andreas Fitzek, Daniel Hein, Stefan Kraxberger, and Clemens Orthacker, *Malware detection by applying knowledge discovery pro-*

cesses to application metadata on the android market (google play), Security and Communication Networks (2013).

- [59] Peter Teufl, Michaela Ferk, Andreas Fitzek, Daniel M. Hein, Stefan Kraxberger, and Clemens Orthacker, *Malware detection by applying knowledge discovery processes to application metadata on the Android Market (Google Play)*., Security and Communication Networks **9** (2016), no. 5, 389–419.
- [60] top10antivirus, *top10antivirus*, <http://antivirus.thetop10sites.com/>.
- [61] tutorials point, *android activities*, http://www.tutorialspoint.com/android/android_activities.htm.
- [62] virustotal, *about virustotal*, <https://www.virustotal.com/en/about/>.
- [63] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos, *Profiledroid: multi-layer profiling of android applications*, Proceedings of the 18th annual international conference on Mobile computing and networking, ACM, 2012, pp. 137–148.
- [64] Primal Wijesekera, Arjun Baokar, Ashkan Hosseini, Serge Egelman, David Wagner, and Konstantin Beznosov, *Android permissions remystified: a field study on contextual integrity*, 24th USENIX Security Symposium (USENIX Security 15), 2015, pp. 499–514.
- [65] Dong-Jie Wu, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu, *Droidmat: Android malware detection through manifest and api calls tracing*, Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on, IEEE, 2012, pp. 62–69.
- [66] Lok Kwong Yan and Heng Yin, *Droidscape: seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis*, Presented as

part of the 21st USENIX Security Symposium (USENIX Security 12), 2012, pp. 569–584.

- [67] Liu Yang, Nader Boushehrinejadmoradi, Pallab Roy, Vinod Ganapathy, and Liviu Iftode, *Short paper: Enhancing users' comprehension of android permissions*, Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (New York, NY, USA), SPSM '12, ACM, 2012, pp. 21–26.
- [68] Min Zheng, Mingshen Sun, and John Lui, *Droid analytics: A signature based analytic system to collect, extract, analyze and associate android malware*, Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on, IEEE, 2013, pp. 163–171.
- [69] Yajin Zhou and Xuxian Jiang, *Dissecting Android malware: Characterization and evolution*, IEEE Symposium on Security and Privacy (SP), IEEE, 2012, pp. 95–109.

Appendix A

Tables in feature chapter

A.1 Size features section

Table A.1: Table of size feature group

Feature name	Description
sizeOfZIPPNG	The total size of compressed image files in an APK. For example the files with suffix: ".png", ".jpg" etc are treated as image files.
sizeOfPNG	The total size of uncompressed image files in APK.
sizeOfZIPXML	The total size of compressed XML files that includes layout files and any resource files who have suffix ".xml" in APK.
sizeOfXML	The total size of uncompressed XML files in APK.
sizeOfZIPDEX	The size of compressed dex file in APK.
sizeOfDEX	The size of uncompressed dex file in APK.
sizeOfZIPZIP	The total size of compressed zip files that have suffix ".zip" in APK.

sizeOfZIP	The total size of uncompressed zip files that have suffix ".zip" in APK.
sizeOfZIPAPK	The total size of compressed apk files that have suffix ".apk" (it is not commonly seen) in APK.
sizeOfAPK	The total size of uncompressed apk files that have suffix ".apk" in APK.
sizeOfZIPARSC	The size of compressed resources.arsc file in APK.
sizeOfARSC	The size of uncompressed resources.arsc file, the resource file in APK.
sizeOfZIPMF	The size of compressed MANIFEST.MF file in APK.
sizeOfMF	The size of uncompressed MANIFEST.MF file in APK.
sizeOfZIPRSA	The size of compressed CERT.RSA files that are the signature file in APK.
sizeOfRSA	The size of uncompressed CERT.RSA files that are the signature file in APK.
sizeOfZIPSF	The size of compressed CERT.SF file in APK.
sizeOfSF	The size of uncompressed CERT.SF file in APK.
sizeOfZIPother	The size of other compressed files that do not belong to any categories listed above.
sizeOfother	The size of other uncompressed files that do not belong to any categories listed above
totalZIPsize	The size of an APK file.
totalsize	The size of an uncompressed APK file.
perOfZIPPNG	The ratio between the size of all the compressed image files and the total APK size.

perOfPNG	The ratio between the size of all uncompressed image files and the total size of the uncompressed APK.
perOfZIPXML	The ratio between the size of all compressed XML files and total size of the APK.
perOfXML	The ratio between the size of all uncompressed XML files and total size of the uncompressed APK.
perOfZIPDEX	The ratio between the size of compressed dex files and the total APK size.
perOfDEX	The ratio between the size of dex file and total size of the uncompressed APK.
perOfZIPZIP	The ratio between the size of all compressed zip files that have ".zip" as the file suffix and the total APK size.
perOfZIP	The ratio between the size of all uncompressed zip files that have ".zip" as the file suffix and total size of the uncompressed APK.
perOfZIPAPK	The ratio between the size of all compressed apk files that have ".apk" as the file suffix and the total APK size.
perOfAPK	The ratio between the size of all uncompressed apk files that have ".apk" as the file suffix and total size of the uncompressed APK.
perOfZIPARSC	The ratio between the size of compressed resources.arsc file and the total APK size.
perOfARSC	The ratio between the size of resources.arsc file and total size of the uncompressed APK.

perOfZIPother	The ratio between the size of compressed files that is not among the categories listed above and the total APK size
perOfother	The ratio between the size of uncompressed files that is not among the categories listed above and the total size of the uncompressed APK.
numOfZVersion	The number of different zip versions used by the files in each APK.
1ZVersion	The version numbers which are used for most of the files.
2ZVersion	The second most used version number of the files in the APK.
perOf1ZVersion	The percentage of the number of files with the most used zip version to the sum of the number of files with the most used zip version(1ZVersion) and the number of files with the second most used zip version(2ZVersion)
RSAZVersion	The zip version of the certification file - CERT.RSA
MFZVersion	The zip version of MANIFEST.MF
SFZVersion	The zip version of CERT.SF

A.2 Timestamp features section

Table A.2: Table of timestamp feature group part 1

Feature name	Description
dexafterzip	The number of dex files that are created after the apk file

dex2mins	The number of dex files that are created in only 2 minutes before the APK file
dex2hours	The number of dex files that are created between 2 minutes to 2 hours before the APK file
dex1day	The number of dex files that are created between 2 hours to 1 day before the creation time of the APK file
dex1month	The number of dex files that are created between 1 day to 1 month before the creation time of the APK file
dexlonger	The number of dex files that are created earlier than 1 month compare to the creation time of the APK file
arscafterzip	The number of arsc files that are created after the apk file
arsc2mins	The number of arsc files that are created in only 2 minutes before the APK file
arsc2hours	The number of arsc files that are created between 2 minutes to 2 hours before the APK file
arsc1day	The number of arsc files that are created between 2 hours to 1 day before the creation time of the APK file
arsc1month	The number of arsc files that are created between 1 day to 1 month before the creation time of the APK file
arsclonger	The number of arsc files that are created earlier than 1 month compare to the creation time of the APK file
MANIFEST_MFafterzip	The number of MANIFEST_MF files that are created after the apk file
MANIFEST_MF2mins	The number of MANIFEST_MF files that are created in only 2 minutes before the APK file

MANIFEST_MF2hours	The number of MANIFEST_MF files that are created between 2 minutes to 2 hours before the APK file
MANIFEST_MF1day	The number of MANIFEST_MF files that are created between 2 hours to 1 day before the creation time of the APK file
MANIFEST_MF1month	The number of MANIFEST_MF files that are created between 1 day to 1 month before the creation time of the APK file
MANIFEST_MFlonger	The number of MANIFEST_MF files that are created earlier than 1 month compare to the creation time of the APK file
CERT_RSAafterzip	The number of CERT_RSA files that are created after the apk file
CERT_RSA2mins	The number of CERT_RSA files that are created in only 2 minutes before the APK file
CERT_RSA2hours	The number of CERT_RSA files that are created between 2 minutes to 2 hours before the APK file
CERT_RSA1day	The number of CERT_RSA files that are created between 2 hours to 1 day before the creation time of the APK file
CERT_RSA1month	The number of CERT_RSA files that are created between 1 day to 1 month before the creation time of the APK file
CERT_RSAlonger	The number of CERT_RSA files that are created earlier than 1 month compare to the creation time of the APK file

CERT_SFAafterzip	The number of CERT_SF files that are created after the apk file
CERT_SF2mins	The number of CERT_SF files that are created in only 2 minutes before the APK file
CERT_SF2hours	The number of CERT_SF files that are created between 2 minutes to 2 hours before the APK file
CERT_SF1day	The number of CERT_SF files that are created between 2 hours to 1 day before the creation time of the APK file
CERT_SF1month	The number of CERT_SF files that are created between 1 day to 1 month before the creation time of the APK file
CERT_SFlonger	The number of CERT_SF files that are created earlier than 1 month compare to the creation time of the APK file
androManifestAafterzip	The number of androManifest files that are created after the apk file
androManifest2mins	The number of androManifest files that are created in only 2 minutes before the APK file
androManifest2hours	The number of androManifest files that are created between 2 minutes to 2 hours before the APK file
androManifest1day	The number of androManifest files that are created between 2 hours to 1 day before the creation time of the APK file

androManifest1month	The number of androManifest files that are created between 1 day to 1 month before the creation time of the APK file
androManifestlonger	The number of androManifest files that are created earlier than 1 month compare to the creation time of the APK file
assetsafterzip	The number of files in the assets folder that are created after the apk file
assets2mins	The number of files in the assets folder that are created in only 2 minutes before the APK file
assets2hours	The number of files in the assets folder that are created between 2 minutes to 2 hours before the APK file
assets1day	The number of files in the assets folder that are created between 2 hours to 1 day before the creation time of the APK file
assets1month	The number of files in the assets folder that are created between 1 day to 1 month before the creation time of the APK file
assetslonger	The number of files in the assets folder that are created earlier than 1 month compare to the creation time of the APK file
libafterzip	The number of files in the lib folder that are created after the apk file
lib2mins	The number of files in the lib folder that are created in only 2 minutes before the APK file

lib2hours	The number of files in the lib folder that are created between 2 minutes to 2 hours before the APK file
lib1day	The number of files in the lib folder that are created between 2 hours to 1 day before the creation time of the APK file
lib1month	The number of files in the lib folder that are created between 1 day to 1 month before the creation time of the APK file
liblonger	The number of files in the lib folder that are created earlier than 1 month compare to the creation time of the APK file
resafterzip	The number of files in the res folder that are created after the apk file
res2mins	The number of files in the res folder that are created in only 2 minutes before the APK file
res2hours	The number of files in the res folder that are created between 2 minutes to 2 hours before the APK file
res1day	The number of files in the res folder that are created between 2 hours to 1 day before the creation time of the APK file
res1month	The number of files in the res folder that are created between 1 day to 1 month before the creation time of the APK file
reslonger	The number of files in the res folder that are created earlier than 1 month compare to the creation time of the APK file

otherafterzip	The number of files do not belong to any necessary folders that are created after the apk file
other2mins	The number of files do not belong to any necessary folders that are created in only 2 minutes before the APK file
other2hours	The number of files do not belong to any necessary folders that are created between 2 minutes to 2 hours before the APK file
other1day	The number of files do not belong to any necessary folders that are created between 2 hours to 1 day before the creation time of the APK file
other1month	The number of files do not belong to any necessary folders that are created between 1 day to 1 month before the creation time of the APK file
otherlonger	The number of files do not belong to any necessary folders that are created earlier than 1 month compare to the creation time of the APK file
fileafterzip	The number of files in APK no matter what types of files they belong to that are created after the apk file
file2mins	The number of files in APK no matter what types of files they belong to that are created in only 2 minutes before the APK file
file2hours	The number of files in APK no matter what types of files they belong to that are created between 2 minutes to 2 hours before the APK file

file1day	The number of files in APK no matter what types of files they belong to that are created between 2 hours to 1 day before the creation time of the APK file
file1month	The number of files in APK no matter what types of files they belong to that are created between 1 day to 1 month before the creation time of the APK file
filelonger	The number of files in APK no matter what types of files they belong to that are created earlier than 1 month compare to the creation time of the APK file
diffassetsfile	The number of different timestamps of the files in the assets folder
totalassetsfile	The total number of files in the assets folder
diffresfile	The number of different timestamps of the files in the res folder
totalresfile	The total number of files in the res folder
difflibfile	The number of different timestamps of the files in the lib folder

Table A.3: TSable of timestamp feature group part 2

Feature name	Description
dexafterzip	The number of dex files which is created after the apk file
$\frac{assetsafterzip}{totalassetsfile}$	Ratio between the number of files in the assets folder that have timestamps greater than APK creation time to the total number of files in the assets folder

$\frac{assets2mins}{totalassetsfile}$	<p>Ratio between the number of files in the assets folder which are created in only 2 minutes before the APK file to the total number of files in the assets folder</p>
$\frac{assets2hours}{totalassetsfile}$	<p>Ratio between the number of files in the assets folder that are created between 2 minutes to 2 hours before the APK file to the total number of files in the assets folder</p>
$\frac{assets1day}{totalassetsfile}$	<p>Ratio between the number of files in the assets folder that are created between 2 hours to 1 day before the creation time of the APK file to the total number of files in the assets folder</p>
$\frac{assets1month}{totalassetsfile}$	<p>Ratio between the number of files in the assets folder that are created between 1 day to 1 month before the creation time of the APK file to the total number of files in the assets folder</p>
$\frac{assetslonger}{totalassetsfile}$	<p>Ratio between the number of files in the assets folder that are created earlier than 1 month compared to the creation time of the APK file to the total number of files in the assets folder</p>
$\frac{diffassetsfile}{totalassetsfile}$	<p>Ratio between the number of different timestamps of the files in the assets folder to the total number of files in the assets folder</p>
$\frac{libafterzip}{totallibfile}$	<p>Ratio between the number of files in the lib folder that have timestamps greater than APK creation time to the total number of files in the lib folder</p>

$\frac{lib2mins}{totallibfile}$	<p>Ratio between the number of files in the lib folder that are created in only 2 minutes before the APK file to the total number of files in the lib folder</p>
$\frac{lib2hours}{totallibfile}$	<p>Ratio between the number of files in the lib folder that are created between 2 minutes to 2 hours before the APK file to the total number of files in the lib folder</p>
$\frac{lib1day}{totallibfile}$	<p>Ratio between the number of files in the lib folder that are created between 2 hours to 1 day before the creation time of the APK file to the total number of files in the lib folder</p>
$\frac{lib1month}{totallibfile}$	<p>Ratio between the number of files in the lib folder that are created between 1 day to 1 month before the creation time of the APK file to the total number of files in the lib folder</p>
$\frac{liblonger}{totallibfile}$	<p>Ratio between the number of files in the lib folder that are created earlier than 1 month compare to the creation time of the APK file to the total number of files in the lib folder</p>
$\frac{difflibfile}{totallibfile}$	<p>Ratio between the number of different timestamps of the files in the lib folder to the total number of files in the lib folder</p>
$\frac{resafterzip}{totalresfile}$	<p>Ratio between the number of files in the res folder that have timestamps greater than APK creation time to the total number of files in the res folder</p>

$\frac{res2mins}{totalresfile}$	<p>Ratio between the number of files in the res folder that are created in only 2 minutes before the APK file to the total number of files in the res folder</p>
$\frac{res2hours}{totalresfile}$	<p>Ratio between the number of files in the res folder that are created between 2 minutes to 2 hours before the APK file to the total number of files in the res folder</p>
$\frac{res1day}{totalresfile}$	<p>Ratio between the number of files in the res folder that are created between 2 hours to 1 day before the creation time of the APK file to the total number of files in the res folder</p>
$\frac{res1month}{totalresfile}$	<p>Ratio between the number of files in the res folder that are created between 1 day to 1 month before the creation time of the APK file to the total number of files in the res folder</p>
$\frac{reslonger}{totalresfile}$	<p>Ratio between the number of files in the res folder that are created earlier than 1 month compared to the creation time of the APK file to the total number of files in the res folder</p>
$\frac{diffresfile}{totalresfile}$	<p>Ratio between the number of different timestamps of the files in the res folder to the total number of files in the res folder</p>
$\frac{othersafterzip}{totalotherfile}$	<p>Ratio between the number of files that do not belong to any necessary folders that are created after the apk file to the total number of files who do not belong to any necessary folders</p>

$\frac{\textit{other2mins}}{\textit{totalother file}}$	<p>Ratio between the number of files that do not belong to any necessary folders which are created in only 2 minutes before the APK file to the total number of files that do not belong to any necessary folders</p>
$\frac{\textit{other2hours}}{\textit{totalother file}}$	<p>Ratio between the number of files that do not belong to any necessary folders that are created between 2 minutes to 2 hours before the APK file to the total number of files that do not belong to any necessary folders</p>
$\frac{\textit{other1day}}{\textit{totalother file}}$	<p>Ratio between the number of files that do not belong to any necessary folders that are created between 2 hours to 1 day before the creation time of the APK file to the total number of files that do not belong to any necessary folders</p>
$\frac{\textit{other1month}}{\textit{totalother file}}$	<p>Ratio between the number of files that do not belong to any necessary folders that are created between 1 day to 1 month before the creation time of the APK file to the total number of files that do not belong to any necessary folders</p>
$\frac{\textit{otherlonger}}{\textit{totalother file}}$	<p>Ratio between the number of files that do not belong to any necessary folders which are created earlier than 1 month compared to the creation time of the APK file to the total number of files do not belong to any necessary folders</p>

$\frac{diff\ other\ file}{total\ other\ file}$	<p>Ratio between the number of different timestamps of the files that do not belong to any necessary folders to the total number of files that do not belong to any necessary folders</p>
$\frac{diff\ assets\ after\ zip}{assets\ after\ zip}$	<p>Ratio between the number of different timestamps of the files created after APK in the assets folder to the total number of files created after APK in the assets folder</p>
$\frac{diff\ assets\ 2\ mins}{assets\ 2\ mins}$	<p>Ratio between the number of different timestamps of the files created 2 minutes before APK in the assets folder to the number of files in the assets folder that are created in only 2 minutes before the APK file</p>
$\frac{diff\ assets\ 2\ hours}{assets\ 2\ hours}$	<p>Ratio between the number of different timestamps of the files created between 2 minutes with 2 hours before APK in the assets folder to the number of files in the assets folder that are created between 2 minutes to 2 hours before the APK file</p>
$\frac{diff\ assets\ 1\ day}{assets\ 1\ day}$	<p>Ratio between the number of different timestamps of the files created between 2 hours with 1 day before APK in The assets folder to the number of files in the assets folder that are created between 2 hours to 1 day before the creation time of the APK file</p>

$\frac{diffassets1month}{assets1month}$	<p>Ratio between the number of different timestamps of the files created between 1 day with 1 month before APK in The assets folder to the number of files in the assets folder that are created between 1 day to 1 month before the creation time of the APK file</p>
$\frac{diffassetslonger}{assetslonger}$	<p>Ratio between the number of different timestamps of the files created more than 1 month before APK in the assets folder to the number of files in the assets folder which is created earlier than 1 month compare to the creation time of the APK file</p>
$\frac{difflibafterzip}{libafterzip}$	<p>Ratio between the number of different timestamps of the files created after APK in the lib folder to the total number of files created after APK in the lib folder</p>
$\frac{difflib2mins}{lib2mins}$	<p>Ratio between the number of different timestamps of the files created 2 minutes before APK in the lib folder to the number of files in the lib folder that are created in only 2 minutes before the APK file</p>
$\frac{difflib2hours}{lib2hours}$	<p>Ratio between the number of different timestamps of the files created between 2 minutes with 2 hours before APK in the lib folder to the number of files in the lib folder that are created between 2 minutes to 2 hours before the APK file</p>

$\frac{difflib1day}{lib1day}$	<p>Ratio between the number of different timestamps of the files created between 2 hours with 1 day before APK in The lib folder to the number of files in the lib folder that are created between 2 hours to 1 day before the creation time of the APK file</p>
$\frac{difflib1month}{lib1month}$	<p>Ratio between the number of different timestamps of the files created between 1 day with 1 month before APK in the lib folder to the number of files in the lib folder that are created between 1 day to 1 month before the creation time of the APK file</p>
$\frac{diffliblonger}{liblonger}$	<p>Ratio between the number of different timestamps of the files created more than 1 month before APK in the lib folder to the number of files in the lib folder that are created earlier than 1 month compare to the creation time of the APK file</p>
$\frac{diffresafterzip}{resafterzip}$	<p>Ratio between the number of different timestamps of the files created after APK in the res folder to the total number of files created after APK in the res folder</p>
$\frac{diffres2mins}{res2mins}$	<p>Ratio between the number of different timestamps of the files created 2 minutes before APK in the res folder to the number of files in the res folder that are created in only 2 minutes before the APK file</p>

$\frac{diffres2hours}{res2hours}$	<p>Ratio between the number of different timestamps of the files created between 2 minutes with 2 hours before APK in the res folder to the number of files in the res folder that are created between 2 minutes to 2 hours before the APK file</p>
$\frac{diffres1day}{res1day}$	<p>Ratio between the number of different timestamps of the files created between 2 hours with 1 day before APK in The res folder to the number of files in the res folder that are created between 2 hours to 1 day before the creation time of the APK file</p>
$\frac{diffres1month}{res1month}$	<p>Ratio between the number of different timestamps of the files created between 1 day with 1 month before APK in the res folder to the number of files in the res folder that are created between 1 day to 1 month before the creation time of the APK file</p>
$\frac{diffreslonger}{reslonger}$	<p>Ratio between the number of different timestamps of the files created more than 1 month before APK in the res folder to the number of files in the res folder that are created earlier than 1 month compare to the creation time of the APK file</p>
$\frac{diffotherafterzip}{otherafterzip}$	<p>Ratio between the number of different timestamps of the files that do not belong to any necessities created after APK to the total number of unnecessary files created after APK</p>

$\frac{diff_{other2mins}}{other2mins}$	<p>Ratio between the number of different timestamps of the files that do not belong to any necessities created 2 minutes before APK to the number of unnecessary files which is created in only 2 minutes before the APK file</p>
$\frac{diff_{other2hours}}{other2hours}$	<p>Ratio between the number of different timestamps of the files that do not belong to any necessities created between 2 minutes with 2 hours before APK to the number of 2 hours before the APK file</p>
$\frac{diff_{other1day}}{other1day}$	<p>Ratio between the number of different timestamps of the files do not belong to any necessities created between 2 hours with 1 day before APK to the number of unnecessary files that are created between 2 hours to 1 day before the creation time of the APK file</p>
$\frac{diff_{other1month}}{other1month}$	<p>Ratio between the number of different timestamps of the files do not belong to any necessities created between 1 day with 1 month before APK to the number of unnecessary files that are created between 1 day to 1 month before the creation time of the APK file</p>
$\frac{diff_{otherlonger}}{otherlonger}$	<p>Ratio between the number of different timestamps of the files do not belong to any necessities created more than 1 month before APK to the number of unnecessary files that are created earlier than 1 month compare to the creation time of the APK file</p>

$\frac{diff_{libfile}}{totalfile}$	Ratio between the number of different timestamps of the files in the lib folder to the number of files in the APK file
$\frac{diff_{resfile}}{totalfile}$	ratio between the number of different timestamps of the files in the res folder to the number of files in the APK file
$\frac{diff_{otherfile}}{totalfile}$	ratio between the number of different timestamps of the unnecessary files in the lib folder to the number of files in the APK file
$\frac{diff_{assetsfile}}{totalfile}$	Ratio between the number of different timestamps of the files in the assets folder to the number of files in the APK file

Table A.4: Table of timestamp feature group part 2

Feature name	Description
dexafterzip	The number of dex files that are created after the apk file
dex2mins	The number of dex files that are created in only 2 minutes before the APK file
dex2hours	The number of dex files that are created between 2 minutes to 2 hours before the APK file
dex1day	The number of dex files that are created between 2 hours to 1 day before the creation time of the APK file

dex1month	The number of dex files that are created between 1 day to 1 month before the creation time of the APK file
dexlonger	The number of dex files that are created earlier than 1 month compare to the creation time of the APK file
arscafterzip	The number of arsc files that are created after the apk file
arsc2mins	The number of arsc files that are created in only 2 minutes before the APK file
arsc2hours	The number of arsc files that are created between 2 minutes to 2 hours before the APK file
arsc1day	The number of arsc files that are created between 2 hours to 1 day before the creation time of the APK file
arsc1month	The number of arsc files that are created between 1 day to 1 month before the creation time of the APK file
arsclonger	The number of arsc files that are created earlier than 1 month compare to the creation time of the APK file
MANIFEST_MFafterzip	The number of MANIFEST_MF files that are created after the apk file
MANIFEST_MF2mins	The number of MANIFEST_MF files that are created in only 2 minutes before the APK file
MANIFEST_MF2hours	The number of MANIFEST_MF files that are created between 2 minutes to 2 hours before the APK file
MANIFEST_MF1day	The number of MANIFEST_MF files that are created between 2 hours to 1 day before the creation time of the APK file

MANIFEST_MF1month	The number of MANIFEST_MF files that are created between 1 day to 1 month before the creation time of the APK file
MANIFEST_MFlonger	The number of MANIFEST_MF files that are created earlier than 1 month compare to the creation time of the APK file
CERT_RSAafterzip	The number of CERT_RSA files that are created after the apk file
CERT_RSA2mins	The number of CERT_RSA files that are created in only 2 minutes before the APK file
CERT_RSA2hours	The number of CERT_RSA files that are created between 2 minutes to 2 hours before the APK file
CERT_RSA1day	The number of CERT_RSA files that are created between 2 hours to 1 day before the creation time of the APK file
CERT_RSA1month	The number of CERT_RSA files that are created between 1 day to 1 month before the creation time of the APK file
CERT_RSAlonger	The number of CERT_RSA files that are created earlier than 1 month compare to the creation time of the APK file
CERT_SFAafterzip	The number of CERT_SF files that are created after the apk file
CERT_SF2mins	The number of CERT_SF files that are created in only 2 minutes before the APK file

CERT_SF2hours	The number of CERT_SF files that are created between 2 minutes to 2 hours before the APK file
CERT_SF1day	The number of CERT_SF files that are created between 2 hours to 1 day before the creation time of the APK file
CERT_SF1month	The number of CERT_SF files that are created between 1 day to 1 month before the creation time of the APK file
CERT_SFlonger	The number of CERT_SF files that are created earlier than 1 month compare to the creation time of the APK file
androManifestAafterzip	The number of androManifest files that are created after the apk file
androManifest2mins	The number of androManifest files that are created in only 2 minutes before the APK file
androManifest2hours	The number of androManifest files that are created between 2 minutes to 2 hours before the APK file
androManifest1day	The number of androManifest files that are created between 2 hours to 1 day before the creation time of the APK file
androManifest1month	The number of androManifest files that are created between 1 day to 1 month before the creation time of the APK file
androManifestlonger	The number of androManifest files that are created earlier than 1 month compare to the creation time of the APK file

assetsafterzip	The number of files in the assets folder that are created after the apk file
assets2mins	The number of files in the assets folder that are created in only 2 minutes before the APK file
assets2hours	The number of files in the assets folder that are created between 2 minutes to 2 hours before the APK file
assets1day	The number of files in the assets folder that are created between 2 hours to 1 day before the creation time of the APK file
assets1month	The number of files in the assets folder that are created between 1 day to 1 month before the creation time of the APK file
assetslonger	The number of files in the assets folder that are created earlier than 1 month compare to the creation time of the APK file
libafterzip	The number of files in the lib folder that are created after the apk file
lib2mins	The number of files in the lib folder that are created in only 2 minutes before the APK file
lib2hours	The number of files in the lib folder that are created between 2 minutes to 2 hours before the APK file
lib1day	The number of files in the lib folder that are created between 2 hours to 1 day before the creation time of the APK file

lib1month	The number of files in the lib folder that are created between 1 day to 1 month before the creation time of the APK file
liblonger	The number of files in the lib folder that are created earlier than 1 month compare to the creation time of the APK file
resafterzip	The number of files in the res folder that are created after the apk file
res2mins	The number of files in the res folder that are created in only 2 minutes before the APK file
res2hours	The number of files in the res folder that are created between 2 minutes to 2 hours before the APK file
res1day	The number of files in the res folder that are created between 2 hours to 1 day before the creation time of the APK file
res1month	The number of files in the res folder that are created between 1 day to 1 month before the creation time of the APK file
reslonger	The number of files in the res folder that are created earlier than 1 month compare to the creation time of the APK file
otherafterzip	The number of files do not belong to any necessary folders that are created after the apk file
other2mins	The number of files do not belong to any necessary folders that are created in only 2 minutes before the APK file

other2hours	The number of files do not belong to any necessary folders that are created between 2 minutes to 2 hours before the APK file
other1day	The number of files do not belong to any necessary folders that are created between 2 hours to 1 day before the creation time of the APK file
other1month	The number of files do not belong to any necessary folders that are created between 1 day to 1 month before the creation time of the APK file
otherlonger	The number of files do not belong to any necessary folders that are created earlier than 1 month compare to the creation time of the APK file
fileafterzip	The number of files in APK no matter what types of files they belong to that are created after the apk file
file2mins	The number of files in APK no matter what types of files they belong to that are created in only 2 minutes before the APK file
file2hours	The number of files in APK no matter what types of files they belong to that are created between 2 minutes to 2 hours before the APK file
file1day	The number of files in APK no matter what types of files they belong to that are created between 2 hours to 1 day before the creation time of the APK file
file1month	The number of files in APK no matter what types of files they belong to that are created between 1 day to 1 month before the creation time of the APK file

filelonger	The number of files in APK no matter what types of files they belong to that are created earlier than 1 month compare to the creation time of the APK file
diffassetsfile	The number of different timestamps of the files in the assets folder
totalassetsfile	The total number of files in the assets folder
diffresfile	The number of different timestamps of the files in the res folder
totalresfile	The total number of files in res folder
difflibfile	The number of different timestamps of the files in the lib folder
totallibfile	The total number of files in the lib folder
$\frac{assetsafterzip}{totalassetsfile}$	Ratio between the number of files in the assets folder that have timestamps greater than APK creation time to the total number of files in the assets folder
$\frac{assets2mins}{totalassetsfile}$	Ratio between the number of files in the assets folder that are created in only 2 minutes before the APK file to the total number of files in the assets folder
$\frac{assets2hours}{totalassetsfile}$	Ratio between the number of files in the assets folder that are created between 2 minutes to 2 hours before the APK file to the total number of files in the assets folder

$\frac{assets1day}{totalassetsfile}$	<p>Ratio between the number of files in the assets folder that are created between 2 hours to 1 day before the creation time of the APK file to the total number of files in the assets folder</p>
$\frac{assets1month}{totalassetsfile}$	<p>Ratio between the number of files in the assets folder that are created between 1 day to 1 month before the creation time of the APK file to the total number of files in the assets folder</p>
$\frac{assetslonger}{totalassetsfile}$	<p>Ratio between the number of files in the assets folder that are created earlier than 1 month compare to the creation time of the APK file to the total number of files in the assets folder</p>
$\frac{diffassetsfile}{totalassetsfile}$	<p>Ratio between the number of different timestamps of the files in the assets folder to the total number of files in the assets folder</p>
$\frac{libafterzip}{totallibfile}$	<p>Ratio between the number of files in the lib folder that have timestamps greater than APK creation time to the total number of files in the lib folder</p>
$\frac{lib2mins}{totallibfile}$	<p>Ratio between the number of files in the lib folder that are created in only 2 minutes before the APK file to the total number of files in the lib folder</p>
$\frac{lib2hours}{totallibfile}$	<p>Ratio between the number of files in the lib folder that are created between 2 minutes to 2 hours before the APK file to the total number of files in the lib folder</p>

$\frac{lib1day}{totallibfile}$	<p>Ratio between the number of files in the lib folder that are created between 2 hours to 1 day before the creation time of the APK file to the total number of files in the lib folder</p>
$\frac{lib1month}{totallibfile}$	<p>Ratio between the number of files in the lib folder that are created between 1 day to 1 month before the creation time of the APK file to the total number of files in the lib folder</p>
$\frac{liblonger}{totallibfile}$	<p>Ratio between the number of files in the lib folder that are created earlier than 1 month compare to the creation time of the APK file to the total number of files in the lib folder</p>
$\frac{difflibfile}{totallibfile}$	<p>Ratio between the number of different timestamps of the files in the lib folder to the total number of files in the lib folder</p>
$\frac{resafterzip}{totalresfile}$	<p>Ratio between the number of files in the res folder that have timestamps greater than APK creation time to the total number of files in the res folder</p>
$\frac{res2mins}{totalresfile}$	<p>Ratio between the number of files in the res folder that are created in only 2 minutes before the APK file to the total number of files in the res folder</p>
$\frac{res2hours}{totalresfile}$	<p>Ratio between the number of files in the res folder that are created between 2 minutes to 2 hours before the APK file to the total number of files in the res folder</p>

$\frac{res1day}{totalresfile}$	<p>Ratio between the number of files in the res folder that are created between 2 hours to 1 day before the creation time of the APK file to the total number of files in the res folder</p>
$\frac{res1month}{totalresfile}$	<p>Ratio between the number of files in the res folder that are created between 1 day to 1 month before the creation time of the APK file to the total number of files in the res folder</p>
$\frac{reslonger}{totalresfile}$	<p>Ratio between the number of files in the res folder that are created earlier than 1 month compare to the creation time of the APK file to the total number of files in the res folder</p>
$\frac{diffresfile}{totalresfile}$	<p>Ratio between the number of different timestamps of the files in the res folder to the total number of files in the res folder</p>
$\frac{otherafterzip}{totalotherfile}$	<p>Ratio between the number of files do not belong to any necessary folders that are created after the apk file to the total number of files do not belong to any necessary folders</p>
$\frac{other2mins}{totalotherfile}$	<p>Ratio between the number of files do not belong to any necessary folders which are created in only 2 minutes before the APK file to the total number of files do not belong to any necessary folders</p>

$\frac{\textit{other2hours}}{\textit{totalotherfile}}$	<p>Ratio between the number of files do not belong to any necessary folders that are created between 2 minutes to 2 hours before the APK file to the total number of files do not belong to any necessary folders</p>
$\frac{\textit{other1day}}{\textit{totalotherfile}}$	<p>Ratio between the number of files do not belong to any necessary folders that are created between 2 hours to 1 day before the creation time of the APK file to the total number of files do not belong to any necessary folders</p>
$\frac{\textit{other1month}}{\textit{totalotherfile}}$	<p>Ratio between the number of files do not belong to any necessary folders that are created between 1 day to 1 month before the creation time of the APK file to the total number of files do not belong to any necessary folders</p>
$\frac{\textit{otherlonger}}{\textit{totalotherfile}}$	<p>Ratio between the number of files do not belong to any necessary folders that are created earlier than 1 month compare to the creation time of the APK file to the total number of files do not belong to any necessary folders</p>
$\frac{\textit{diffotherfile}}{\textit{totalotherfile}}$	<p>Ratio between the number of different timestamps of the files do not belong to any necessary folders to the total number of files do not belong to any necessary folders</p>
$\frac{\textit{diffassetsafterzip}}{\textit{assetsafterzip}}$	<p>Ratio between the number of different timestamps of the files created after APK in the assets folder to the total number of files created after APK in the assets folder</p>

$\frac{diffassets2mins}{assets2mins}$	<p>Ratio between the number of different timestamps of the files created 2 minutes before APK in the assets folder to the number of files in the assets folder that is created in only 2 minutes before the APK file</p>
$\frac{diffassets2hours}{assets2hours}$	<p>Ratio between the number of different timestamps of the files created between 2 minutes with 2 hours before APK in the assets folder to the number of files in the assets folder that are created between 2 minutes to 2 hours before the APK file</p>
$\frac{diffassets1day}{assets1day}$	<p>Ratio between the number of different timestamps of the files created between 2 hours with 1 day before APK in the assets folder to the number of files in the assets folder that are created between 2 hours to 1 day before the creation time of the APK file</p>
$\frac{diffassets1month}{assets1month}$	<p>Ratio between the number of different timestamps of the files created between 1 day with 1 month before APK in the assets folder to the number of files in the assets folder that are created between 1 day to 1 month before the creation time of the APK file</p>
$\frac{diffassetslonger}{assetslonger}$	<p>Ratio between the number of different timestamps of the files created more than 1 month before APK in the assets folder to the number of files in the assets folder that are created earlier than 1 month compare to the creation time of the APK file</p>

$\frac{difflibafterzip}{libafterzip}$	<p>Ratio between the number of different timestamps of the files created after APK in the lib folder to the total number of files created after APK in the lib folder</p>
$\frac{difflib2mins}{lib2mins}$	<p>Ratio between the number of different timestamps of the files created 2 minutes before APK in the lib folder to the number of files in the lib folder that are created in only 2 minutes before the APK file</p>
$\frac{difflib2hours}{lib2hours}$	<p>Ratio between the number of different timestamps of the files created between 2 minutes with 2 hours before APK in the lib folder to the number of files in the lib folder that are created between 2 minutes to 2 hours before the APK file</p>
$\frac{difflib1day}{lib1day}$	<p>Ratio between the number of different timestamps of the files created between 2 hours with 1 day before APK in the lib folder to the number of files in the lib folder that are created between 2 hours to 1 day before the creation time of the APK file</p>
$\frac{difflib1month}{lib1month}$	<p>Ratio between the number of different timestamps of the files created between 1 day with 1 month before APK in the lib folder to the number of files in the lib folder that are created between 1 day to 1 month before the creation time of the APK file</p>

$\frac{diffliblonger}{liblonger}$	<p>Ratio between the number of different timestamps of the files created more than 1 month before APK in the lib folder to the number of files in the lib folder that are created earlier than 1 month compare to the creation time of the APK file</p>
$\frac{diffresafterzip}{resafterzip}$	<p>Ratio between the number of different timestamps of the files created after APK in the res folder to the total number of files created after APK in the res folder</p>
$\frac{diffres2mins}{res2mins}$	<p>Ratio between the number of different timestamps of the files created 2 minutes before APK in the res folder to the number of files in the res folder that are created in only 2 minutes before the APK file</p>
$\frac{diffres2hours}{res2hours}$	<p>Ratio between the number of different timestamps of the files created between 2 minutes with 2 hours before APK in the res folder to the number of files in the res folder that are created between 2 minutes to 2 hours before the APK file</p>
$\frac{diffres1day}{res1day}$	<p>Ratio between the number of different timestamps of the files created between 2 hours with 1 day before APK in the res folder to the number of files in the res folder that are created between 2 hours to 1 day before the creation time of the APK file</p>

$\frac{diffres1month}{res1month}$	<p>Ratio between the number of different timestamps of the files created between 1 day with 1 month before APK in the res folder to the number of files in the res folder that are created between 1 day to 1 month before the creation time of the APK file</p>
$\frac{diffreslonger}{reslonger}$	<p>Ratio between the number of different timestamps of the files created more than 1 month before APK in the res folder to the number of files in the res folder that are created earlier than 1 month compare to the creation time of the APK file</p>
$\frac{diffotherafterzip}{otherafterzip}$	<p>Ratio between the number of different timestamps of the files do not belong to any necessities created after APK to the total number of unnecessary files created after APK</p>
$\frac{diffother2mins}{other2mins}$	<p>Ratio between the number of different timestamps of the files do not belong to any necessities created 2 minutes before APK to the number of unnecessary files that are created in only 2 minutes before the APK file</p>
$\frac{diffother2hours}{other2hours}$	<p>Ratio between the number of different timestamps of the files do not belong to any necessities created between 2 minutes with 2 hours before APK to the number of unnecessary files that are created between 2 minutes to 2 hours before the APK file</p>

$\frac{diff\ other\ 1\ day}{other\ 1\ day}$	<p>Ratio between the number of different timestamps of the files do not belong to any necessities created between 2 hours with 1 day before APK to the number of unnecessary files that are created between 2 hours to 1 day before the creation time of the APK file</p>
$\frac{diff\ other\ 1\ month}{other\ 1\ month}$	<p>Ratio between the number of different timestamps of the files do not belong to any necessities created between 1 day with 1 month before APK to the number of unnecessary files that are created between 1 day to 1 month before the creation time of the APK file</p>
$\frac{diff\ other\ longer}{other\ longer}$	<p>Ratio between the number of different timestamps of the files do not belong to any necessities created more than 1 month before APK to the number of unnecessary files that are created earlier than 1 month compare to the creation time of the APK file</p>
$\frac{diff\ lib\ file}{total\ file}$	<p>Ratio between the number of different timestamps of the files in the lib folder to the number of files in the APK file</p>
$\frac{diff\ res\ file}{total\ file}$	<p>Ratio between the number of different timestamps of the files in the res folder to the number of files in the APK file</p>
$\frac{diff\ other\ file}{total\ file}$	<p>Ratio between the number of different timestamps of the unnecessary files in the lib folder to the number of files in the APK file</p>

$\frac{diffassetsfile}{totalfile}$	Ratio between the number of different timestamps of the files in the assets folder to the number of files in the APK file
afterMANIFEST_MF	The number of files with greater timestamps compare to MANIFEST_MF
afterCERT_RSA	The number of files with greater timestamps compare to the CERT_RSA file
afterCERT_SF	The number of files with greater timestamps compare to the CERT_SF file
$\frac{afterMANIFEST_MF}{totalfile}$	The ratio of afterMANIFEST_MF to total number of files
$\frac{afterCERT_RSA}{totalfile}$	The ratio of afterCERT_RSA to total number of files
$\frac{afterCERT_SF}{totalfile}$	The ratio of afterCERT_SF to the total number of files

A.3 Manifest features section

In the table following, external elements stand for the elements attached from external libraries (like ads libraries) and internal elements stand for the elements implemented by authors (name of those elements starts with . or the package name of that APK)

Table A.6: Table of manifest feature group

Feature name	Description
rate0	Ratio of the number of external elements in manifest file with benign ratio 0.0 to the number of all elements in manifest file
rate20	Ratio of the number of external elements in manifest file with benign ratio from 0.0 to 0.2 to the number of all elements in manifest file
rate40	Ratio of the number of external elements in manifest file with benign ratio from 0.2 to 0.4 to the number of all elements in manifest file
rate60	Ratio of the number of external elements in manifest file with benign ratio from 0.4 to 0.6 to the number of all elements in manifest file
rate80	Ratio of the number of external elements in manifest file with benign ratio from 0.6 to 0.8 to the number of all elements in manifest file
rate100	Ratio of the number of external elements in manifest file with benign ratio from 0.8 to 1 to the number of all elements in manifest file

unknownRate	Ratio of the number of external elements in manifest file with unknown benign ratio to the number of all elements in manifest file
underPackageRate	Ratio of the number of internal elements in manifest file to the number of all elements in manifest file
To0	Number of external elements in manifest file with benign ratio 0.0
To20	Number of external elements in manifest file with benign ratio from 0.0 to 0.2
To40	Number of external elements in manifest file with benign ratio from 0.2 to 0.4
To60	Number of external elements in manifest file with benign ratio from 0.4 to 0.6
To80	Number of external elements in manifest file with benign ratio from 0.6 to 0.8
To100	Number of external elements in manifest file with benign ratio from 0.8 to 1.0
unknown	Number of external elements in manifest file with unknown benign ratio
underPackage	Number of internal elements in manifest file
actionNum	Number of actions in manifest file
activityNum	Number of activities in manifest file
meta_dataNum	Number of meta-datas in manifest file
permissionNum	Number of permissions in manifest file
providerNum	Number of providers in manifest file
receiverNum	Number of receivers in manifest file

serviceNum	Number of services in manifest file
uses_featureNum	Number of uses_features in manifest file
uses_permissionNum	Number of uses_permissions in manifest file
actionratio	Ratio of the number of actions in manifest file to the number of all elements in manifest file
activityratio	Ratio of the number of activities in manifest file to the number of all elements in manifest file
meta_dataratio	Ratio of the number of meta-datas in manifest file to the number of all elements in manifest file
permissionratio	Ratio of the number of permissions in manifest file to the number of all elements in manifest file
providerratio	Ratio of the number of providers in manifest file to the number of all elements in manifest file
receiverratio	Ratio of the number of receivers in manifest file to the number of all elements in manifest file
serviceratio	Ratio of the number of services in manifest file to the number of all elements in manifest file
uses_featureratio	Ratio of the number of uses_features in manifest file to the number of all elements in manifest file
uses_permissionratio	Ratio of the number of uses_permissions in manifest file to the number of all elements in manifest file
outAction	Number of developer defined actions
outPermission	Number of developer defined permissions
outActionRate	Ratio of the number of developer defined actions to the number of all actions in manifest file

outPermissionRate	Ratio of the number of developer defined permissions to the number of all permissions in manifest file
implicitIntent	Number of implicit intents
explicitIntent	Number of explicit intents
otherIntent	Number of intents including category tags
allIntent	Number of all intents in manifest file
implicitIntentRatio	Ratio of the number of implicit intents to the total number of intents in manifest file
explicitIntentRatio	Ratio of the number of explicit intents to the total number of intents in manifest file
otherIntentRatio	Ratio of the number of outIntents to the total number of intents in manifest file
priorityNormal	Number of normal priorities
priorityAbnormal	Number of abnormal priorities
priorityAll	Number of all priorities
priorityNormalRatio	Ratio of the number of normal priorities to the number of all priorities
priorityAbnormalRatio	Ratio of the number of abnormal priorities to the number of all priorities
highestPriority	The highest priority value in manifest file
lowestPriority	The lowest priority value in manifest file
uses-feature_0	Number of uses-features with benign ratio 0.0 in manifest file
uses-feature_0_ratio	Ratio of the number of uses-features with benign ratio 0.0 to the total number of elements with benign ratio 0.0

permission_0	Number of permissions with benign ratio 0.0 in manifest file
permission_0_ratio	Ratio of the number of permissions with benign ratio 0.0 to the total number of elements with benign ratio 0.0
meta-data_0	Number of meta-datas with benign ratio 0.0 in manifest file
meta-data_0_ratio	Ratio of the number of meta-datas with benign ratio 0.0 to the total number of elements with benign ratio 0.0
action_0	Number of actions with benign ratio 0.0 in manifest file
action_0_ratio	Ratio of the number of actions with benign ratio 0.0 to the total number of elements with benign ratio 0.0
provider_0	Number of providers with benign ratio 0.0 in manifest file
provider_0_ratio	Ratio of the number of providers with benign ratio 0.0 to the total number of elements with benign ratio 0.0
uses-permission_0	Number of uses-permissions with benign ratio 0.0 in manifest file
uses-permission_0_ratio	Ratio of the number of uses-permissions with benign ratio 0.0 to the total number of elements with benign ratio 0.0
activity_0	Number of activities with benign ratio 0.0 in manifest file
activity_0_ratio	Ratio of the number of activities with benign ratio 0.0 to the total number of elements with benign ratio 0.0
service_0	Number of services with benign ratio 0.0 in manifest file

service_0_ratio	Ratio of the number of services with benign ratio 0.0 to the total number of elements with benign ratio 0.0
receiver_0	Number of receivers with benign ratio 0.0 in manifest file
receiver_0_ratio	Ratio of the number of receivers with benign ratio 0.0 to the total number of elements with benign ratio 0.0
meta-data_20	Number of meta-datas with benign ratio from 0.0 to 0.2 in manifest file
meta-data_20_ratio	Ratio of the number of meta-datas with benign ratio from 0.0 to 0.2 to the total number of elements with benign ratio from 0.0 to 0.2
permission_20	Number of permissions with benign ratio from 0.0 to 0.2 in manifest file
permission_20_ratio	Ratio of the number of permissions with benign ratio from 0.0 to 0.2 to the total number of elements with benign ratio from 0.0 to 0.2
service_20	Number of services with benign ratio from 0.0 to 0.2 in manifest file
service_20_ratio	Ratio of the number of services with benign ratio from 0.0 to 0.2 to the total number of elements with benign ratio from 0.0 to 0.2
activity_20	Number of activities with benign ratio from 0.0 to 0.2 in manifest file
activity_20_ratio	Ratio of the number of activities with benign ratio from 0.0 to 0.2 to the total number of elements with benign ratio from 0.0 to 0.2

receiver_20	Number of receivers with benign ratio from 0.0 to 0.2 in manifest file
receiver_20_ratio	Ratio of the number of receivers with benign ratio from 0.0 to 0.2 to the total number of elements with benign ratio from 0.0 to 0.2
provider_20	Number of providers with benign ratio from 0.0 to 0.2 in manifest file
provider_20_ratio	Ratio of the number of providers with benign ratio from 0.0 to 0.2 to the total number of elements with benign ratio from 0.0 to 0.2
uses-permission_20	Number of uses-permissions with benign ratio from 0.0 to 0.2 in manifest file
uses-permission_20_ratio	Ratio of the number of uses-permissions with benign ratio from 0.0 to 0.2 to the total number of elements with benign ratio from 0.0 to 0.2
action_20	Number of actions with benign ratio from 0.0 to 0.2 in manifest file
action_20_ratio	Ratio of the number of actions with benign ratio from 0.0 to 0.2 to the total number of elements with benign ratio from 0.0 to 0.2
uses-feature_20	Number of uses-features with benign ratio from 0.0 to 0.2 in manifest file
uses-feature_20_ratio	Ratio of the number of uses-features with benign ratio from 0.0 to 0.2 to the total number of elements with benign ratio from 0.0 to 0.2

uses-permission_40	Number of uses-permissions with benign ratio from 0.2 to 0.4 in manifest file
uses-permission_40_ratio	Ratio of the number of uses-permissions with benign ratio from 0.2 to 0.4 to the total number of elements with benign ratio from 0.2 to 0.4
receiver_40	Number of receivers with benign ratio from 0.2 to 0.4 in manifest file
receiver_40_ratio	Ratio of the number of receivers with benign ratio from 0.2 to 0.4 to the total number of elements with benign ratio from 0.2 to 0.4
meta-data_40	Number of meta-datas with benign ratio from 0.2 to 0.4 in manifest file
meta-data_40_ratio	Ratio of the number of meta-datas with benign ratio from 0.2 to 0.4 to the total number of elements with benign ratio from 0.2 to 0.4
action_40	Number of actions with benign ratio from 0.2 to 0.4 in manifest file
action_40_ratio	Ratio of the number of actions with benign ratio from 0.2 to 0.4 to the total number of elements with benign ratio from 0.2 to 0.4
permission_40	Number of permissions with benign ratio from 0.2 to 0.4 in manifest file
permission_40_ratio	Ratio of the number of permissions with benign ratio from 0.2 to 0.4 to the total number of elements with benign ratio from 0.2 to 0.4

service_40	Number of services with benign ratio from 0.2 to 0.4 in manifest file
service_40_ratio	Ratio of the number of services with benign ratio from 0.2 to 0.4 to the total number of elements with benign ratio from 0.2 to 0.4
uses-feature_40	Number of uses-features with benign ratio from 0.2 to 0.4 in manifest file
uses-feature_40_ratio	Ratio of the number of uses-features with benign ratio from 0.2 to 0.4 to the total number of elements with benign ratio from 0.2 to 0.4
activity_40	Number of activities with benign ratio from 0.2 to 0.4 in manifest file
activity_40_ratio	Ratio of the number of activities with benign ratio from 0.2 to 0.4 to the total number of elements with benign ratio from 0.2 to 0.4
provider_40	Number of providers with benign ratio from 0.2 to 0.4 in manifest file
provider_40_ratio	Ratio of the number of providers with benign ratio from 0.2 to 0.4 to the total number of elements with benign ratio from 0.2 to 0.4
uses-feature_60	Number of uses-features with benign ratio from 0.4 to 0.6 in manifest file
uses-feature_60_ratio	Ratio of the number of uses-features with benign ratio from 0.4 to 0.6 to the total number of elements with benign ratio from 0.4 to 0.6

action_60	Number of actions with benign ratio from 0.4 to 0.6 in manifest file
action_60_ratio	Ratio of the number of actions with benign ratio from 0.4 to 0.6 to the total number of elements with benign ratio from 0.4 to 0.6
provider_60	Number of providers with benign ratio from 0.4 to 0.6 in manifest file
provider_60_ratio	Ratio of the number of providers with benign ratio from 0.4 to 0.6 to the total number of elements with benign ratio from 0.4 to 0.6
permission_60	Number of permissions with benign ratio from 0.4 to 0.6 in manifest file
permission_60_ratio	Ratio of the number of permissions with benign ratio from 0.4 to 0.6 to the total number of elements with benign ratio from 0.4 to 0.6
receiver_60	Number of receivers with benign ratio from 0.4 to 0.6 in manifest file
receiver_60_ratio	Ratio of the number of receivers with benign ratio from 0.4 to 0.6 to the total number of elements with benign ratio from 0.4 to 0.6
uses-permission_60	Number of uses-permissions with benign ratio from 0.4 to 0.6 in manifest file
uses-permission_60_ratio	Ratio of the number of uses-permissions with benign ratio from 0.4 to 0.6 to the total number of elements with benign ratio from 0.4 to 0.6

activity_60	Number of activities with benign ratio from 0.4 to 0.6 in manifest file
activity_60_ratio	Ratio of the number of activities with benign ratio from 0.4 to 0.6 to the total number of elements with benign ratio from 0.4 to 0.6
service_60	Number of services with benign ratio from 0.4 to 0.6 in manifest file
service_60_ratio	Ratio of the number of services with benign ratio from 0.4 to 0.6 to the total number of elements with benign ratio from 0.4 to 0.6
meta-data_60	Number of meta-datas with benign ratio from 0.4 to 0.6 in manifest file
meta-data_60_ratio	Ratio of the number of meta-datas with benign ratio from 0.4 to 0.6 to the total number of elements with benign ratio from 0.4 to 0.6
activity_80	Number of activities with benign ratio from 0.6 to 0.8 in manifest file
activity_80_ratio	Ratio of the number of activities with benign ratio from 0.6 to 0.8 to the total number of elements with benign ratio from 0.6 to 0.8
meta-data_80	Number of meta-datas with benign ratio from 0.6 to 0.8 in manifest file
meta-data_80_ratio	Ratio of the number of meta-datas with benign ratio from 0.6 to 0.8 to the total number of elements with benign ratio from 0.6 to 0.8

uses-feature_80	Number of uses-features with benign ratio from 0.6 to 0.8 in manifest file
uses-feature_80_ratio	Ratio of the number of uses-features with benign ratio from 0.6 to 0.8 to the total number of elements with benign ratio from 0.6 to 0.8
uses-permission_80	Number of uses-permissions with benign ratio from 0.6 to 0.8 in manifest file
uses-permission_80_ratio	Ratio of the number of uses-permissions with benign ratio from 0.6 to 0.8 to the total number of elements with benign ratio from 0.6 to 0.8
provider_80	Number of providers with benign ratio from 0.6 to 0.8 in manifest file
provider_80_ratio	Ratio of the number of providers with benign ratio from 0.6 to 0.8 to the total number of elements with benign ratio from 0.6 to 0.8
receiver_80	Number of receivers with benign ratio from 0.6 to 0.8 in manifest file
receiver_80_ratio	Ratio of the number of receivers with benign ratio from 0.6 to 0.8 to the total number of elements with benign ratio from 0.6 to 0.8
action_80	Number of actions with benign ratio from 0.6 to 0.8 in manifest file
action_80_ratio	Ratio of the number of actions with benign ratio from 0.6 to 0.8 to the total number of elements with benign ratio from 0.6 to 0.8

permission_80	Number of permissions with benign ratio from 0.6 to 0.8 in manifest file
permission_80_ratio	Ratio of the number of permissions with benign ratio from 0.6 to 0.8 to the total number of elements with benign ratio from 0.6 to 0.8
service_80	Number of services with benign ratio from 0.6 to 0.8 in manifest file
service_80_ratio	Ratio of the number of services with benign ratio from 0.6 to 0.8 to the total number of elements with benign ratio from 0.6 to 0.8
meta-data_100	Number of meta-datas with benign ratio from 0.8 to 1.0 in manifest file
meta-data_100_ratio	Ratio of the number of meta-datas with benign ratio from 0.8 to 1.0 to the total number of elements with benign ratio from 0.8 to 1.0
activity_100	Number of activities with benign ratio from 0.8 to 1.0 in manifest file
activity_100_ratio	Ratio of the number of activities with benign ratio from 0.8 to 1.0 to the total number of elements with benign ratio from 0.8 to 1.0
action_100	Number of actions with benign ratio from 0.8 to 1.0 in manifest file
action_100_ratio	Ratio of the number of actions with benign ratio from 0.8 to 1.0 to the total number of elements with benign ratio from 0.8 to 1.0

uses-feature_100	Number of uses-features with benign ratio from 0.8 to 1.0 in manifest file
uses-feature_100_ratio	Ratio of the number of uses-features with benign ratio from 0.8 to 1.0 to the total number of elements with benign ratio from 0.8 to 1.0
service_100	Number of services with benign ratio from 0.8 to 1.0 in manifest file
service_100_ratio	Ratio of the number of services with benign ratio from 0.8 to 1.0 to the total number of elements with benign ratio from 0.8 to 1.0
receiver_100	Number of receivers with benign ratio from 0.8 to 1.0 in manifest file
receiver_100_ratio	Ratio of the number of receivers with benign ratio from 0.8 to 1.0 to the total number of elements with benign ratio from 0.8 to 1.0
provider_100	Number of providers with benign ratio from 0.8 to 1.0 in manifest file
provider_100_ratio	Ratio of the number of providers with benign ratio from 0.8 to 1.0 to the total number of elements with benign ratio from 0.8 to 1.0
uses-permission_100	Number of uses-permissions with benign ratio from 0.8 to 1.0 in manifest file
uses-permission_100_ratio	Ratio of the number of uses-permissions with benign ratio from 0.8 to 1.0 to the total number of elements with benign ratio from 0.8 to 1.0

permission_100	Number of permissions with benign ratio from 0.8 to 1.0 in manifest file
permission_100_ratio	Ratio of the number of permissions with benign ratio from 0.8 to 1.0 to the total number of elements with benign ratio from 0.8 to 1.0
uses-permission_unknown	Number of uses-permissions with unknown benign ratio
uses-permission_unknown_ratio	Ratio of the number of uses-permissions with unknown benign ratio to the total number of elements with unknown benign ratio
action_unknown	Number of actions with unknown benign ratio
action_unknown_ratio	Ratio of the number of actions with unknown benign ratio to the total number of elements with unknown benign ratio
activity_unknown	Number of activities with unknown benign ratio
activity_unknown_ratio	Ratio of the number of activities with unknown benign ratio to the total number of elements with unknown benign ratio
receiver_unknown	Number of receivers with unknown benign ratio
receiver_unknown_ratio	Ratio of the number of receivers with unknown benign ratio to the total number of elements with unknown benign ratio
provider_unknown	Number of providers with unknown benign ratio
provider_unknown_ratio	Ratio of the number of providers with unknown benign ratio to the total number of elements with unknown benign ratio

permission_unknown	Number of permissions with unknown benign ratio
permission_unknown_ratio	Ratio of the number of permissions with unknown benign ratio to the total number of elements with unknown benign ratio
meta-data_unknown	Number of meta-datas with unknown benign ratio
meta-data_unknown_ratio	Ratio of the number of meta-datas with unknown benign ratio to the total number of elements with unknown benign ratio
service_unknown	Number of services with unknown benign ratio
service_unknown_ratio	Ratio of the number of services with unknown benign ratio to the total number of elements with unknown benign ratio
uses-feature_unknown	Number of uses-features with unknown benign ratio
uses-feature_unknown_ratio	Ratio of the number of uses-features with unknown benign ratio to the total number of elements with unknown benign ratio
PHONE	Number of dangerous permission belong to PHONE category
PHONE_ratio	Ratio of the number of dangerous permissions belong to PHONE category to the total number of permissions
MICROPHONE	Number of dangerous permission belong to MICROPHONE category
MICROPHONE_ratio	Ratio of the number of dangerous permissions belong to MICROPHONE category to the total number of permissions

CAMERA	Number of dangerous permission belong to CAMERA category
CAMERA_ratio	Ratio of the number of dangerous permissions belong to CAMERA category to the total number of permissions
LOCATION	Number of dangerous permission belong to LOCATION category
LOCATION_ratio	Ratio of the number of dangerous permissions belong to LOCATION category to the total number of permissions
CONTACTS	Number of dangerous permission belong to CONTACTS category
CONTACTS_ratio	Ratio of the number of dangerous permissions belong to CONTACTS category to the total number of permissions
STORAGE	Number of dangerous permission belong to STORAGE category
STORAGE_ratio	Ratio of the number of dangerous permissions belong to STORAGE category to the total number of permissions
CALENDAR	Number of dangerous permission belong to CALENDAR category
CALENDAR_ratio	Ratio of the number of dangerous permissions belong to CALENDAR category to the total number of permissions
SMS	Number of dangerous permission belong to SMS category
SMS_ratio	Ratio of the number of dangerous permissions belong to SMS category to the total number of permissions

SENSORS	Number of dangerous permission belong to SENSORS category
SENSORS_ratio	Ratio of the number of dangerous permissions belong to SENSORS category to the total number of permissions

$\frac{\textit{assetsafterzip}}{\textit{totalassetsfile}}$	Ratio between the number of files in the assets folder that have timestamps greater than APK creation time to the total number of files in the assets folder
$\frac{\textit{assets2mins}}{\textit{totalassetsfile}}$	Ratio between the number of files in the assets folder that are created in only 2 minutes before the APK file to the total number of files in the assets folder
$\frac{\textit{assets2hours}}{\textit{totalassetsfile}}$	Ratio between the number of files in the assets folder that are created between 2 minutes to 2 hours before the APK file to the total number of files in the assets folder
$\frac{\textit{assets1day}}{\textit{totalassetsfile}}$	Ratio between the number of files in the assets folder that are created between 2 hours to 1 day before the creation time of the APK file to the total number of files in the assets folder
$\frac{\textit{assets1month}}{\textit{totalassetsfile}}$	Ratio between the number of files in the assets folder that are created between 1 day to 1 month before the creation time of the APK file to the total number of files in the assets folder
$\frac{\textit{assetslonger}}{\textit{totalassetsfile}}$	Ratio between the number of files in the assets folder that are created earlier than 1 month compare to the creation time of the APK file to the total number of files in the assets folder
$\frac{\textit{diffassetsfile}}{\textit{totalassetsfile}}$	Ratio between the number of different timestamps of the files in the assets folder to the total number of files in the assets folder

Table A.5: Ratio of files in the assets folder

Appendix B

Figures for structures of apps made by different tools

The content in red boxes is the unique pattern for the apps made by that tool

```

Zip file size: 5336339 bytes, number of entries: 529
-rw---- 2.0 fat 5180 bx defN 15-Apr-24 15:21 AndroidManifest.xml
-rw---- 2.0 fat 396 bl defN 15-Apr-24 15:21 res/anim/abc_fade_in.xml
-rw---- 2.0 fat 396 bl defN 15-Apr-24 15:21 res/anim/abc_fade_out.xml
-rw---- 2.0 fat 400 bl defN 15-Apr-24 15:21 res/anim/abc_slide_in_bottom.xml
-rw---- 2.0 fat 400 bl defN 15-Apr-24 15:21 res/anim/abc_slide_in_top.xml
-rw---- 2.0 fat 400 bl defN 15-Apr-24 15:21 res/anim/abc_slide_out_bottom.xml
-rw---- 2.0 fat 400 bl defN 15-Apr-24 15:21 res/anim/abc_slide_out_top.xml
-rw---- 2.0 fat 780 bl defN 15-Apr-24 15:21 res/anim/arrow.xml
-rw---- 2.0 fat 424 bl defN 15-Apr-24 15:21 res/anim/push_bottom_in.xml
-rw---- 2.0 fat 424 bl defN 15-Apr-24 15:21 res/anim/push_bottom_out.xml
-rw---- 2.0 fat 428 bl defN 15-Apr-24 15:21 res/anim/push_left_in.xml
-rw---- 2.0 fat 428 bl defN 15-Apr-24 15:21 res/anim/push_left_left.xml
-rw---- 2.0 fat 428 bl defN 15-Apr-24 15:21 res/anim/push_left_out.xml
-rw---- 2.0 fat 428 bl defN 15-Apr-24 15:21 res/anim/push_left_right.xml
-rw---- 2.0 fat 428 bl defN 15-Apr-24 15:21 res/anim/push_up_down.xml
-rw---- 2.0 fat 428 bl defN 15-Apr-24 15:21 res/anim/push_up_in.xml
-rw---- 2.0 fat 428 bl defN 15-Apr-24 15:21 res/anim/push_up_out.xml
-rw---- 2.0 fat 428 bl defN 15-Apr-24 15:21 res/anim/push_up_up.xml
-rw---- 2.0 fat 596 bl defN 15-Apr-24 15:21 res/color/abc_search_url_text_holo.xml

Zip file size: 22200383 bytes, number of entries: 1354
-rw---- 2.0 fat 9460 bx defN 15-Jan-28 13:18 AndroidManifest.xml
-rw---- 2.0 fat 31484 bl defN 15-Jan-28 13:18 assets/GarfieldDiner.ttf
-rw---- 2.0 fat 29948 bl defN 15-Jan-28 13:18 assets/Helvetica Bold.ttf
-rw---- 2.0 fat 11178 bl defN 15-Jan-28 13:18 assets/arial12.fnt
-rw---- 1.0 fat 8452 b- stor 15-Jan-28 11:12 assets/arial12.png
-rw---- 2.0 fat 11313 bl defN 15-Jan-28 13:18 assets/arial18.fnt
-rw---- 1.0 fat 12712 b- stor 15-Jan-28 11:12 assets/arial18.png
-rw---- 1.0 fat 6203 b- stor 15-Jan-28 11:12 assets/fps_images.png
-rw---- 1.0 fat 8748 b- stor 15-Jan-28 11:12 assets/hd/images/Challeng_banner.png
-rw---- 1.0 fat 4069 b- stor 15-Jan-28 11:12 assets/hd/images/ChallengaAchieved_A.png
-rw---- 1.0 fat 3923 b- stor 15-Jan-28 11:12 assets/hd/images/ChallengaAchieved_B.png
-rw---- 1.0 fat 4063 b- stor 15-Jan-28 11:12 assets/hd/images/ChallengaAchieved_C.png
-rw---- 1.0 fat 3482 b- stor 15-Jan-28 11:12 assets/hd/images/ChallengaAchieved_S.png
-rw---- 1.0 fat 4740 b- stor 15-Jan-28 11:12 assets/hd/images/ChallengaFail_PopUp.png
-rw---- 1.0 fat 1874 b- stor 15-Jan-28 11:12 assets/hd/images/MissionComplete_block.png
-rw---- 1.0 fat 3816 b- stor 15-Jan-28 11:12 assets/hd/images/Mission_Complete.png
-rw---- 1.0 fat 2409 b- stor 15-Jan-28 11:12 assets/hd/images/UI_GameMenu_next_day.png
-rw---- 1.0 fat 1782 b- stor 15-Jan-28 11:12 assets/hd/images/UI_btn_wallpaper.png

Zip file size: 1519146 bytes, number of entries: 285
-rw---- 2.0 fat 16756 bx defN 15-Mar-26 13:37 AndroidManifest.xml
-rw---- 2.0 fat 396 bl defN 15-Mar-26 13:37 res/anim/fade_in.xml
-rw---- 2.0 fat 396 bl defN 15-Mar-26 13:37 res/anim/fade_out.xml
-rw---- 2.0 fat 400 bl defN 15-Mar-26 13:37 res/anim/hold.xml
-rw---- 2.0 fat 428 bl defN 15-Mar-26 13:37 res/anim/push_left_in.xml
-rw---- 2.0 fat 428 bl defN 15-Mar-26 13:37 res/anim/push_left_out.xml
-rw---- 2.0 fat 428 bl defN 15-Mar-26 13:37 res/anim/push_right_in.xml
-rw---- 2.0 fat 428 bl defN 15-Mar-26 13:37 res/anim/push_right_out.xml
-rw---- 2.0 fat 440 bl defN 15-Mar-26 13:37 res/anim/umeng_fb_slide_in_from_left.xml
-rw---- 2.0 fat 440 bl defN 15-Mar-26 13:37 res/anim/umeng_fb_slide_in_from_right.xml
-rw---- 2.0 fat 440 bl defN 15-Mar-26 13:37 res/anim/umeng_fb_slide_out_from_left.xml
-rw---- 2.0 fat 440 bl defN 15-Mar-26 13:37 res/anim/umeng_fb_slide_out_from_right.xml
-rw---- 1.0 fat 524 b- stor 15-Mar-26 13:36 res/drawable-hdpi-v4/arrow_down.png
-rw---- 1.0 fat 511 b- stor 15-Mar-26 13:36 res/drawable-hdpi-v4/arrow_right.png
-rw---- 1.0 fat 496 b- stor 15-Mar-26 13:36 res/drawable-hdpi-v4/arrow_up.png
-rw---- 1.0 fat 1311 b- stor 15-Mar-26 13:36 res/drawable-hdpi-v4/bg_mycourse_full_starbg.png
-rw---- 1.0 fat 1165 b- stor 15-Mar-26 13:36 res/drawable-hdpi-v4/bg_mycourse_normal_starbg.png
-rw---- 1.0 fat 1191 b- stor 15-Mar-26 13:36 res/drawable-hdpi-v4/bg_mycourse_lstbg.png
-rw---- 1.0 fat 1688 b- stor 15-Mar-26 13:36 res/drawable-hdpi-v4/bg_project_classification.png
-rw---- 1.0 fat 309 b- stor 15-Mar-26 13:36 res/drawable-hdpi-v4/bg_project_title.png
-rw---- 1.0 fat 2749 b- stor 15-Mar-26 13:36 res/drawable-hdpi-v4/btn_feedback.png

```

Figure B.1: Android Studio & Eclipse app zipinfo top lines

```

-rw---- 1.0 fat 132728 b- stor 15-Apr-24 15:21 resources.arsc
-rw---- 2.0 fat 1805140 bl defN 15-Apr-24 15:21 classes.dex
-rw---- 2.0 fat 53510 bl defN 15-Apr-24 15:21 META-INF/MANIFEST.MF
-rw---- 2.0 fat 53563 bl defN 15-Apr-24 15:21 META-INF/CERT.SF
-rw---- 2.0 fat 1085 bl defN 15-Apr-24 15:21 META-INF/CERT.RSA
529 files, 6704738 bytes uncompressed, 5242709 bytes compressed: 21.8%

-rw---- 2.0 fat 980 bl defN 15-Jan-28 13:18 js/closebutton.html
-rw---- 2.0 fat 54871 bl defN 15-Jan-28 13:18 js/mraid.js
-rw---- 2.0 fat 122795 bl defN 15-Jan-28 13:18 META-INF/MANIFEST.MF
-rw---- 2.0 fat 122824 bl defN 15-Jan-28 13:18 META-INF/CERT.SF
-rw---- 2.0 fat 724 bl defN 15-Jan-28 13:18 META-INF/CERT.RSA
1354 files, 29705216 bytes uncompressed, 21988252 bytes compressed: 26.0%

-rw---- 2.0 fat 4162 bl defN 15-Mar-26 13:37 assets/drawable-xxhdpi/lc_com_sina_weibo_sdk_close.png
-rw---- 2.0 fat 120244 bl defN 15-Mar-26 13:37 lib/armeabi/libjpush.so
-rw---- 2.0 fat 120244 bl defN 15-Mar-26 13:37 lib/armeabi-v7a/libjpush.so
-rw---- 2.0 fat 25226 bl defN 15-Mar-26 13:37 META-INF/MANIFEST.MF
-rw---- 2.0 fat 25279 bl defN 15-Mar-26 13:37 META-INF/CERT.SF
-rw---- 2.0 fat 1359 bl defN 15-Mar-26 13:37 META-INF/CERT.RSA
285 files, 2640117 bytes uncompressed, 1474180 bytes compressed: 44.2%

```

Figure B.2: Android Studio & Eclipse app zipinfo bottom lines

```

Zip file size: 19285162 bytes, number of entries: 143
-rw-r--r-- 2.3 unx 2080 tx defN 13-Jan-29 11:36 assets/META-INF/AIR/application.xml
-rw-r--r-- 2.3 unx 3346 tx stor 13-Jan-29 11:36 res/drawable-ldpi/icon.png
-rw-r--r-- 2.3 unx 5374 tx stor 13-Jan-29 11:36 res/drawable-mdpi/icon.png
-rw-r--r-- 2.3 unx 10958 tx stor 13-Jan-29 11:36 res/drawable-hdpi/icon.png
-rw-r--r-- 2.3 unx 269420 tx defN 13-Jan-29 11:36 classes.dex
-rw-r--r-- 2.3 unx 3285 tx stor 13-Jan-29 11:36 res/drawable/air_72px_mobile_eula.png
-rw-r--r-- 2.3 unx 991 tx stor 13-Jan-29 11:36 res/drawable/mp_warning_32x32_n.png

Zip file size: 34843303 bytes, number of entries: 77
-rw-r--r-- 2.3 unx 3482 tx defN 14-Jan-02 16:31 assets/META-INF/AIR/application.xml
-rw-r--r-- 2.3 unx 2154 tx stor 14-Jan-02 16:31 res/drawable-ldpi/icon.png
-rw-r--r-- 2.3 unx 3159 tx stor 14-Jan-02 16:31 res/drawable-mdpi/icon.png
-rw-r--r-- 2.3 unx 5332 tx stor 14-Jan-02 16:31 res/drawable-hdpi/icon.png
-rw-r--r-- 2.3 unx 70666 tx defN 14-Jan-02 16:31 res/drawable-xhdpi/ouya_icon.png
-rw-r--r-- 2.3 unx 7903 tx defN 14-Jan-02 16:31 res/drawable-xhdpi/icon.png
-rw-r--r-- 2.3 unx 13312 tx defN 14-Jan-02 16:31 res/drawable-xxhdpi/icon.png
-rw-r--r-- 2.3 unx 436336 tx defN 14-Jan-02 16:31 classes.dex
-rw-r--r-- 2.3 unx 3285 tx stor 14-Jan-02 16:31 res/drawable/air_72px_mobile_eula.png
-rw-r--r-- 2.3 unx 13375 tx defN 14-Jan-02 16:31 res/drawable/loading.png

Zip file size: 16692310 bytes, number of entries: 739
-rw-r--r-- 2.3 unx 12476 tx defN 15-Jul-15 00:41 assets/META-INF/AIR/application.xml
-rw-r--r-- 2.3 unx 2688 tx stor 15-Jul-15 00:41 res/drawable-ldpi/icon.png
-rw-r--r-- 2.3 unx 3843 tx stor 15-Jul-15 00:41 res/drawable-mdpi/icon.png
-rw-r--r-- 2.3 unx 6645 tx stor 15-Jul-15 00:41 res/drawable-hdpi/icon.png
-rw-r--r-- 2.3 unx 15824 tx defN 15-Jul-15 00:41 res/drawable-xhdpi/ouya_icon.png
-rw-r--r-- 2.3 unx 13378 tx stor 15-Jul-15 00:41 res/drawable-xhdpi/icon.png
-rw-r--r-- 2.3 unx 15824 tx defN 15-Jul-15 00:41 res/drawable-xxhdpi/icon.png
-rw-r--r-- 2.3 unx 15824 tx defN 15-Jul-15 00:41 res/drawable-xxxhdpi/icon.png
-rw-r--r-- 2.3 unx 2545864 tx defN 15-Jul-15 00:41 classes.dex

```

Figure B.3: Adobe air APKs zipinfo top lines

```

-rw-r--r-- 2.3 unx 47721 tx defN 12-Nov-02 08:35 assets/com/greensock/TimelineMax.as
-rw-r--r-- 2.3 unx 780 tx defN 12-Nov-02 08:35 assets/com/greensock/TweenAlign.as
-rw-r--r-- 2.3 unx 41319 tx defN 12-Nov-02 08:35 assets/com/greensock/TweenLite.as
-rw-r--r-- 2.3 unx 73108 tx defN 12-Nov-02 08:35 assets/com/greensock/TweenMax.as
-rw-r--r-- 2.3 unx 21769 tx defN 12-Nov-02 08:35 assets/com/greensock/TweenNano.as
-rw-r--r-- 2.3 unx 9830151 tx stor 13-Jan-29 11:36 assets/killerscape.swf
-rw-r--r-- 2.3 unx 13372 tx defN 13-Jan-29 11:36 META-INF/MANIFEST.MF
-rw-r--r-- 2.3 unx 13425 tx defN 13-Jan-29 11:36 META-INF/CERT.SF
-rw-r--r-- 2.3 unx 938 tx defN 13-Jan-29 11:36 META-INF/CERT.RSA
143 files, 29172847 bytes uncompressed, 19260485 bytes compressed: 34.0%

-rw-r--r-- 2.3 unx 5332 tx stor 13-Nov-14 19:39 assets/icons/icon72.png
-rw-r--r-- 2.3 unx 5858 tx stor 13-Nov-14 19:45 assets/icons/icon76.png
-rw-r--r-- 2.3 unx 6266 tx stor 13-Nov-14 19:45 assets/icons/icon80.png
-rw-r--r-- 2.3 unx 7903 tx defN 13-Nov-14 19:39 assets/icons/icon96.png
-rw-r--r-- 2.3 unx 6432 tx defN 14-Jan-02 16:32 META-INF/MANIFEST.MF
-rw-r--r-- 2.3 unx 6485 tx defN 14-Jan-02 16:32 META-INF/CERT.SF
-rw-r--r-- 2.3 unx 907 tx defN 14-Jan-02 16:32 META-INF/CERT.RSA
77 files, 44857520 bytes uncompressed, 34831392 bytes compressed: 22.4%

-rw-r--r-- 2.3 unx 16762 tx defN 15-Jul-15 00:40 assets/themes/glossy/sounds/piece_move_1.mp3
-rw-r--r-- 2.3 unx 12640 tx defN 15-Jul-15 00:40 assets/themes/glossy/sounds/piece_move_2.mp3
-rw-r--r-- 2.3 unx 4699 tx defN 15-Jul-15 00:40 assets/themes/glossy/sounds/piece_select.mp3
-rw-r--r-- 2.3 unx 19326 tx defN 15-Jul-15 00:40 assets/themes/glossy/sounds/piece_unmove.mp3
-rw-r--r-- 2.3 unx 2959527 tx defN 15-Jul-15 00:41 assets/Xiangqi.swf
-rw-r--r-- 2.3 unx 103226 tx defN 15-Jul-15 00:42 META-INF/MANIFEST.MF
-rw-r--r-- 2.3 unx 103279 tx defN 15-Jul-15 00:42 META-INF/CERT.SF
-rw-r--r-- 2.3 unx 925 tx defN 15-Jul-15 00:42 META-INF/CERT.RSA
739 files, 28979759 bytes uncompressed, 16501071 bytes compressed: 43.1%

```

Figure B.4: Adobe air APKs zipinfo bottom lines

```

Zip file size: 10296285 bytes, number of entries: 1733
-rw---- 1.0 fat 161891 b-defN 15-Dec-04 16:34 META-INF/MANIFEST.MF
-rw---- 1.0 fat 162012 b-defN 15-Dec-04 16:34 META-INF/CERT.SF
-rw---- 1.0 fat 1339 b-defN 15-Dec-04 16:34 META-INF/CERT.RSA
-rw---- 1.0 fat 5808 b-defN 15-Dec-04 16:34 AndroidManifest.xml
-rw---- 1.0 fat 1396292 b-defN 15-Dec-04 16:34 assets/cludades.txt
-rw---- 1.0 fat 158604 b-defN 15-Dec-04 16:34 assets/fonts/Roboto-Regular.ttf
-rw---- 1.0 fat 1336 b-defN 15-Dec-04 16:34 assets/juntos1.txt
-rw---- 1.0 fat 71034 b-defN 15-Dec-04 16:34 assets/juntos10.txt
-rw---- 1.0 fat 1812 b-defN 15-Dec-04 16:34 assets/juntos100.txt
-rw---- 1.0 fat 1540 b-defN 15-Dec-04 16:34 assets/juntos101.txt
-rw---- 1.0 fat 1320 b-defN 15-Dec-04 16:34 assets/juntos102.txt
-rw---- 1.0 fat 666 b-defN 15-Dec-04 16:34 assets/juntos103.txt
-rw---- 1.0 fat 1379 b-defN 15-Dec-04 16:34 assets/juntos104.txt
-rw---- 1.0 fat 1028 b-defN 15-Dec-04 16:34 assets/juntos105.txt
-rw---- 1.0 fat 780 b-defN 15-Dec-04 16:34 assets/juntos106.txt
-rw---- 1.0 fat 1054 b-defN 15-Dec-04 16:34 assets/juntos107.txt
-rw---- 1.0 fat 488 b-defN 15-Dec-04 16:34 assets/juntos108.txt

Zip file size: 5682868 bytes, number of entries: 590
-rw---- 1.0 fat 57122 b-defN 15-May-30 12:01 META-INF/MANIFEST.MF
-rw---- 1.0 fat 57243 b-defN 15-May-30 12:01 META-INF/CERT.SF
-rw---- 1.0 fat 1329 b-defN 15-May-30 12:01 META-INF/CERT.RSA
-rw---- 1.0 fat 100 b-defN 15-May-30 12:01 com/bytebolt/drumloopsfree/lang.properties
-rw---- 1.0 fat 3345 b-defN 15-May-30 12:01 com/bytebolt/musicbase/Midi/Example.xml
-rw---- 1.0 fat 59 b-defN 15-May-30 12:01 .readme
-rw---- 1.0 fat 231896 b-defN 15-May-30 12:01 lib/armeabi/libbass.so
-rw---- 1.0 fat 51380 b-defN 15-May-30 12:01 lib/armeabi/libbassflac.so
-rw---- 1.0 fat 120076 b-defN 15-May-30 12:01 lib/armeabi/libbassmidi.so
-rw---- 1.0 fat 46288 b-defN 15-May-30 12:01 lib/armeabi/libbasssw.so
-rw---- 1.0 fat 215524 b-defN 15-May-30 12:01 lib/armeabi-v7a/libbass.so
-rw---- 1.0 fat 47296 b-defN 15-May-30 12:01 lib/armeabi-v7a/libbassflac.so
-rw---- 1.0 fat 87320 b-defN 15-May-30 12:01 lib/armeabi-v7a/libbassmidi.so
-rw---- 1.0 fat 46300 b-defN 15-May-30 12:01 lib/armeabi-v7a/libbasssw.so
-rw---- 1.0 fat 293604 b-defN 15-May-30 12:01 lib/x86/libbass.so
-rw---- 1.0 fat 84224 b-defN 15-May-30 12:01 lib/x86/libbassflac.so
-rw---- 1.0 fat 107692 b-defN 15-May-30 12:01 lib/x86/libbassmidi.so
-rw---- 1.0 fat 79100 b-defN 15-May-30 12:01 lib/x86/libbasssw.so
-rw---- 1.0 fat 5480 b-defN 15-May-30 12:01 AndroidManifest.xml

Zip file size: 6278045 bytes, number of entries: 1073
-rw---- 1.0 fat 105174 b-defN 15-Sep-09 19:28 META-INF/MANIFEST.MF
-rw---- 1.0 fat 105295 b-defN 15-Sep-09 19:28 META-INF/CERT.SF
-rw---- 1.0 fat 1136 b-defN 15-Sep-09 19:28 META-INF/CERT.RSA
-rw---- 1.0 fat 59 b-defN 15-Sep-09 19:28 .readme
-rw---- 1.0 fat 71764 b-defN 15-Sep-09 19:28 com/google/api/client/googleapis/google.jks
-rw---- 1.0 fat 89 b-defN 15-Sep-09 19:28 manifest
-rw---- 1.0 fat 867 b-stor 15-Sep-09 19:28 drawable/copyright.png
-rw---- 1.0 fat 2994 b-stor 15-Sep-09 19:28 drawable/detail_button.png
-rw---- 1.0 fat 2166 b-stor 15-Sep-09 19:28 drawable/info_box.png
-rw---- 1.0 fat 1390 b-stor 15-Sep-09 19:28 drawable/info_box_tail.png
-rw---- 1.0 fat 4655 b-stor 15-Sep-09 19:28 drawable/map_pin.png
-rw---- 1.0 fat 4441 b-stor 15-Sep-09 19:28 drawable/map_pin_custom.png
-rw---- 1.0 fat 4323 b-stor 15-Sep-09 19:28 drawable/map_pin_red.png
-rw---- 1.0 fat 1308 b-stor 15-Sep-09 19:28 drawable/map_pin_shadow.png
-rw---- 1.0 fat 3072 b-stor 15-Sep-09 19:28 drawable/map_present.png
-rw---- 1.0 fat 2728 b-stor 15-Sep-09 19:28 drawable/map_present_direction.png
-rw---- 1.0 fat 1084 b-stor 15-Sep-09 19:28 drawable/map_present_tracking.png

```

Figure B.5: Dot42 APKs zipinfo partial lines

```

Zip file size: 50934755 bytes, number of entries: 1302
-rw---- 1.0 fat 21576 bx-stor 14-Nov-17 17:14 assets/data/audio/abilities/aby_abilities_blues_loop.mp3
-rw---- 1.0 fat 8044 b-stor 14-Nov-17 17:14 assets/data/audio/abilities/aby_abilities_blues_start.mp3
-rw---- 1.0 fat 7862 b-stor 14-Nov-17 17:14 assets/data/audio/abilities/aby_abilities_blues_stop.mp3
-rw---- 1.0 fat 18284 b-stor 14-Nov-17 17:14 assets/data/audio/abilities/aby_abilities_bomb.mp3
-rw---- 1.0 fat 32913 b-stor 14-Nov-17 17:14 assets/data/audio/abilities/aby_abilities_bubbles_loop.mp3
-rw---- 1.0 fat 6033 b-stor 14-Nov-17 17:14 assets/data/audio/abilities/aby_abilities_bubbles_start.mp3
-rw---- 1.0 fat 15624 b-stor 14-Nov-17 17:14 assets/data/audio/abilities/aby_abilities_bubbles_stop.mp3
-rw---- 1.0 fat 19656 b-stor 14-Nov-17 17:14 assets/data/audio/abilities/aby_abilities_chuck_loop.mp3
-rw---- 1.0 fat 6552 b-stor 14-Nov-17 17:14 assets/data/audio/abilities/aby_abilities_chuck_start.mp3
-rw---- 1.0 fat 5796 b-stor 14-Nov-17 17:14 assets/data/audio/abilities/aby_abilities_chuck_stop.mp3

Zip file size: 1828824 bytes, number of entries: 135
-rw---- 1.0 fat 57633 bx-stor 13-Sep-03 11:07 res/drawable/a.jpg
-rw---- 2.0 fat 700 bl-defN 13-Sep-03 11:21 res/drawable/adapterback.xml
-rw---- 1.0 fat 237 b-stor 13-Sep-03 11:07 res/drawable/adapterback_w.png
-rw---- 1.0 fat 6283 b-stor 13-Sep-03 11:07 res/drawable/adapterback_y.png
-rw---- 1.0 fat 312 b-stor 13-Sep-03 11:07 res/drawable/arrow.png
-rw---- 1.0 fat 59529 b-stor 13-Sep-03 11:07 res/drawable/b.jpg
-rw---- 1.0 fat 2227 b-stor 13-Sep-03 11:07 res/drawable/back.png
-rw---- 1.0 fat 433 b-stor 13-Sep-03 11:07 res/drawable/bottombtn_w.png
-rw---- 1.0 fat 777 b-stor 13-Sep-03 11:07 res/drawable/bottombtn_y.png
-rw---- 2.0 fat 840 bl-defN 13-Sep-03 11:21 res/drawable/btn_back_selector.xml
-rw---- 2.0 fat 544 bl-defN 13-Sep-03 11:21 res/drawable/btn_top_pressed.xml

Zip file size: 23371591 bytes, number of entries: 459
-rw---- 2.0 fat 39424 bx-defN 13-Jun-02 11:20 assets/bin/data/Managed/Assembly-CSharp-ffrstopass.dll
-rw---- 2.0 fat 73216 bl-defN 13-Jun-02 11:20 assets/bin/data/Managed/Assembly-CSharp.dll
-rw---- 2.0 fat 17408 bl-defN 13-Jun-02 11:20 assets/bin/data/Managed/P31RestKit.dll
-rw---- 2.0 fat 7168 bl-defN 13-Jun-02 11:20 assets/bin/data/Managed/System.Core.dll
-rw---- 2.0 fat 13312 bl-defN 13-Jun-02 11:20 assets/bin/data/Managed/System.dll
-rw---- 2.0 fat 195584 bl-defN 13-Jun-02 11:20 assets/bin/data/Managed/UnityEngine.dll
-rw---- 2.0 fat 1347584 bl-defN 13-Jun-02 11:20 assets/bin/data/Managed/mscorlib.dll
-rw---- 2.0 fat 4096 bl-defN 13-Jun-02 11:20 assets/bin/data/00000000000000000000000000000000
-rw---- 2.0 fat 4096 bl-defN 13-Jun-02 11:20 assets/bin/data/18a14d6bfb3447c0a9b80f5db401318
-rw---- 2.0 fat 4096 bl-defN 13-Jun-02 11:20 assets/bin/data/1d7ebc638d3b840478cf7297e9aa6237

```

Figure B.6: Unknown tool

Glossary (if any)

start writing here.

Vita

Candidate's full name: Yan Li

University attended (with dates and degrees obtained): University of New Brunswick

Publications:

Conference Presentations: